Stochastic Knowledge Representations and Machine Learning Strategies for Biological Sequence Analysis

Hiroshi Mamitsuka

Abstract

We establish novel stochastic knowledge representations and new machine learning strategies to solve four important problems in the field of molecular biology. In particular, we focus on the problem of predicting protein structures, e.g. α-helices, β-sheets and typical structural motifs, which has been regarded as a principal theme in this field.

A common feature of the four methods is that each builds a stochastic model to represent the target objective in the respective problem setting, and the methods are expected to be robust against errors or noise which are liable to occur in biological databases obtained through rather manual and complicated biochemical experiments. It should be emphasized that our algorithms for training each of the stochastic models from given examples have a sufficient degree of originality (particularly in terms of computer science) relative to conventional algorithms for each of the problems involved, and that we evaluate our methods in computational experiments and in the experiments performed, more favorable performance can be obtained with them than with other existing algorithms.

Following a brief introduction, this thesis begins in Chapter 2 by introducing the problems in molecular biology and the techniques of machine learning that are dealt with in this thesis. Specifically, we concentrate on the problem of predicting protein structures in detail. We emphasize that all of our learning methods are based on the minimum description length (MDL) principle, the use of which is described in Chapters 3 and 4, and on maximum likelihood estimation which is the basis of all the algorithms outside of MDL learning described in this thesis.

In Chapter 3, we address the problem of predicting protein α-helix regions, which are one of the major secondary structures and is believed to be formed from its local properties. For this problem, we define a *stochastic rule with finite partitioning* to represent an amino acid distribution at each residue position in an α-helix. We then establish a strategy based on the MDL principle to optimize the structure of the stochastic rule. In other words, we obtain optimal clustering of amino acid types at a position. To make the optimization possible, we use the sequences whose three-dimensional structures are unknown, to enhance the amount of available data and greatly improve the prediction accuracy of our method. Among the experiments we conducted was a large-scale one involving data on more than five thousand amino acids, and the results obtained show that the method we developed achieved 81% prediction accuracy. This exceeded the 75% accuracy obtained with Qian and Sejnowski's (QS) method for the same data and was on the same level as the results obtained with Rost and Sander's (RS) method which was widely considered to be the best secondary structure prediction method. One of the merits of our method is that it can provide comprehensible rules of α-helices while QS and RS cannot.

In Chapter 4, we propose a new method for representing inter-residue relations in motifs, which are structural or functional key portions hidden in amino acid sequences. We define a *probabilistic network with finite partitionings* in which an arc represents an inter-residue relation between nodes corresponding to residue positions. Based on the MDL principle and a greedy-search from given examples, we establish an efficient learning algorithm which automatically constructs a near-optimal probabilistic network. Our experiments using an actual protein motif show that our method built a probabilistic network with finite partitionings, in which each inter-residue relation corresponded to an actual bio-chemical feature peculiar to the motif. Moreover, this network, which provides visible inter-residue relations, had the same level of motif classification accuracy as a feed-forward type neural network trained by an ordinary learning approach, which network however cannot provide any visible information.

In Chapter 5, we propose a new method for learning a *hidden Markov model*, which have been used as a model for representing multiple aligned sequences of a certain functional or structural class. The conventional algorithm of hidden Markov models, called the Baum-Welch algorithm, has the disadvantage of not always being able to provide sufficient discrimination ability because the algorithm uses only the sequences belonging to the class of interest, and not those that do not. On the other hand, we establish a new efficient learning algorithm which uses not only the sequences belonging to the class, but also sequences that do not, called *negative examples.* This allows us to enhance the discrimination ability of the Baum-Welch algorithm. Furthermore, with our algorithm, hidden Markov models can be applied to data in which each example has a label. In other words, our method realizes a supervised learning of hidden Markov models. In our experiments, we used actual amino acid sequences consisting of both positive and negative examples of an existing motif in order to evaluate the discrimination ability of our method and of three other existing methods including the Baum-Welch algorithm. Experimental results show that for the dataset in question, our method greatly reduced discrimination errors as compared to the other two methods.

In Chapter 6, we propose a new method for the problem of predicting the location of β-sheets, which along with α-helices, are one of the major secondary structures. The difficulty of this problem is that the dependencies of β-strands forming a β-sheet in the amino-acid sequence are unbounded. To cope with this difficulty, we define a new family of stochastic tree grammars which we call a *stochastic ranked node rewriting grammar*, which is powerful enough to capture such unbounded dependencies. Furthermore, we establish a new learning algorithm for the tree grammars, and add a number of significant modifications to the grammars and the algorithm. In our experiments, we conducted a number of experiments with data in which no test sequence has more than 25% pairwise sequence similarity to any training protein. Experimental results show that our method captured the long distance dependency in β-sheet regions, thus providing positive evidence for the potential of our method as a tool for scientific discovery that allows us to discover unnoticed structural similarity in proteins having no or little sequence similarity.

In Chapter 7, we give concluding remarks and mention some possible future work related to enhance the performance of our current methods.

Acknowledgments

The author would like to express his deepest gratitude to his thesis supervisor Prof. Satoru Miyano of the University of Tokyo for his valuable guidance and advice, which was greatly helpful in completing this thesis. The author would also like to thank the members of his thesis commitee: Prof. Kentaro Shimizu, Prof. Hiroshi Imai and Prof. Tatsuya Akutsu of the University of Tokyo and Prof. Tomoyuki Higuchi of the Institute of Statistical Mathematics for their valuable comments and suggestions.

The author would like to express his sincere gratitude to Dr. Naoki Abe and Dr. Kenji Yamanishi of C&C Media Research Laboratories, NEC Corporation. They have offered a number of advice and suggestions since he joined NEC Corporation, and their early guidance instilled in the author confidence in his research ability. Chapters 3 and 6 of this thesis are fruitful results produced by joint work with them.

The author also thanks Prof. Akihiko Konagaya of JAIST. It is Prof. Konagaya who first introduced the author to the field of computational biology (or genetic information processing), while he was with NEC Corporation.

The research described in this thesis was conducted at C&C Media Research Labs, NEC Corporation. The author thanks Atsuyoshi Nakamura, Li Hang and Jun-ichi Takeuchi of the "Machine Learning" group of C&C Media Research Labs, NEC Corpotation for their helpful advice and suggestions. The author also thanks Mr. Katsuhiro Nakamura, Mr. Tomoyuki Fujita and Dr. Shun Doi of NEC Corporation for their constant support and encouragement.

Special thanks also go to Dr. Ken-ichi Takada of NIS, who implemented the parallel parsing algorithm of stochastic tree grammars in Chapter 6.

Publication Notes

Chapter 3 is based on three papers which appeared in the journal *Computer Applications in the Biosciences* (Mamitsuka & Yamanishi, 1995), in the proceedings of the *26th Hawaii International Conference on System Sciences* (Mamitsuka & Yamanishi, 1993) and in the proceedings of the *Third Workshop on Algorithmic Learning Theory* (Mamitsuka & Yamanishi, 1992). All of these represent joint work participated in by Dr. Kenji Yamanishi of NEC Corporation.

Chapter 4 is based on two papers which appeared in the journal *Computer Applications in the Biosciences* (Mamitsuka, 1995) and appeared in the proceedings of *Genome Informatics Workshop IV* (Mamitsuka, 1993).

Chapter 5 is based on three papers which appeared in the *Journal of Computational Biology* (Mamitsuka, 1996) , in the journal *Proteins: Structure, Function and Genetics* (Mamitsuka, 1998) and in the proceedings of the *First International Conference on Computational Molecular Biology* (Mamitsuka, 1997).

Chapter 6 is based on five papers which appeared in the journal *Machine Learning* (Abe & Mamitsuka, 1997), in the proceedings of the *11th International Conference on Machine Learning* (Abe & Mamitsuka, 1994), in the proceedings of the *Second International Conference on Intelligent Systems for Molecular Biology* (Mamitsuka & Abe, 1994a), in the proceedings of *Genome Informatics Workshop V* (Mamitsuka & Abe, 1994b) and in the proceedings of the *15th International Conference on Machine Learning* (Abe & Mamitsuka, 1998). All of these represent joint work participated in by Dr. Naoki Abe of NEC Corporation.

Contents

Chapter 1

Introduction

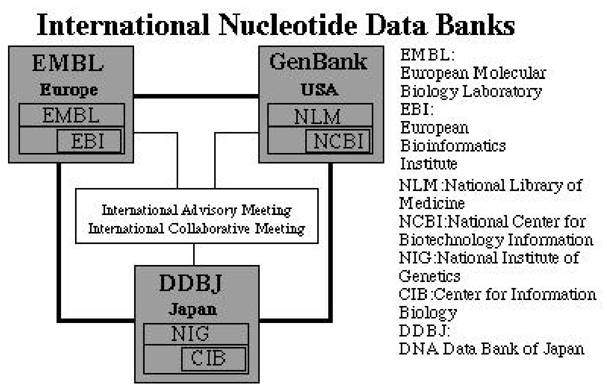
Computational Needs in Molecular Biology

A number of world-wide genome sequencing projects for various types of living organisms including human beings have been promoted over the world since the early 1990s. With the advent of recent newly developed technologies in genetic engineering, these projects have rapidly accumulated an enormous amount of genetic sequence data for some living organisms, e.g. *Arabidopsis thaliana* (Meinke et al., 1998) and *Caenorhabditis elegans* (The *C.elegans* consortium, 1998), during the last these few years. Surprisingly, one report (Pennisi, 1999) predicts that most of the entire human genome, which corresponds to twenty-three human chromosome pairs, i.e. nearly three billion bases, will be sequenced at an accuracy of 99% by the spring of 2000. This is considerably earlier than the initial target date of 2005 which had previously estimated. This amount of information on the human genome, corresponding to 750 megabytes of digital information (Olson, 1995), is said to be equal to the amount of information printed in a major newspaper over a period of almost twenty years.

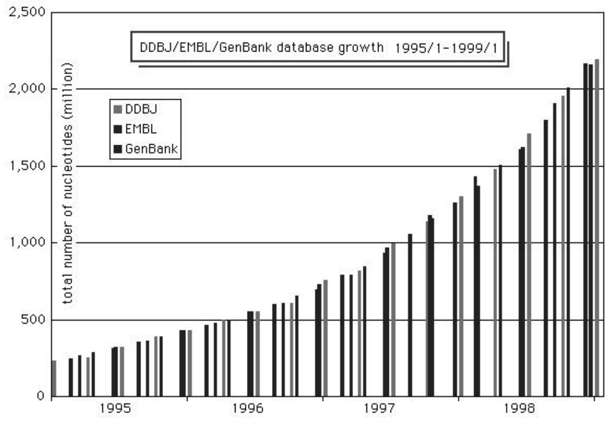
Actually, sequence databases to which newly sequenced genetic data submitted have taken a quantum leap in size these last few years. To date, there exist three major nucleotide sequence databases, i.e. DDBJ (DNA Data Bank of Japan) (Sugawara, Miyazaki, Gojobori, & Tateno, 1999), GenBank (Benson, Boguski, Lipman, Ostell, Ouelette, Rapp, & Wheeler, 1999), and EMBL (European Molecular Biology Laboratory Nucleotide Sequence Databank) (Stoesser, Tuli, Lopez, & Sterk, 1999) in Japan, USA and Europe, respectively, which databases complement each other as shown in Figure 1.1 (a). Figure 1.1 (b) which shows the accumulated number of nucleotides in these databases during the past few years, indicates that for all three, the number of annual nucleotides has drastically increased every year and that the total number of nucleotides surpassed twenty billion in 1998. According to the DDBJ, the total number of the nucleotides in these databases may be in one hundred of billions in the near future, given the accelerating pace at which sequencing data is being accumulated.

The genetic sequence data being determined or which will be determined in this way includes information not only on each component in living organisms but also information regarding the timing with which components are produced from their genes, and even information about the interactive functions between the components, and thus we can say that the sequence information is a compressed cipher to explicate all living phenomena in this world. In other words, we can say that the sequence data is the most fundamental information in molecular biology.

Under these circumstances, to elucidate a variety of living phenomena and further to regulate them, we need to extract higher order information from the accumulated fundamental information in molecular biology. A typical example of extracting higher order information from the sequence data is to predict the 3-dimensional structure of a protein whose fundamental information has already been



(*a*)



(*b*)

Figure 1.1: Nucleotide sequence databases and their statistics
from http://www.ddbj.nig.ac.jp

determined, the ultimate aims being to modify (a) the protein's structure and (b) the structure of a ligand bound to the protein. Those aims are often referred to as 'protein design' and 'drug design', respectively.

To automatically extract such types of higher order information from an enormous amount of fundamental data, the techniques with which the data can be processed with computers are strongly required in not only biological science but also a number of other related fields such as medical science, pharmacy, and agriculture (Casari et al., 1994; Taubes, 1996)

Since the late 1980s, a large number of computer scientists have started to seriously tackle the problems being created in molecular biology with the rapid increase in biological sequence data. As a result of this, a new scientific field referred to as 'computational molecular biology' or 'bioinformatics' has been wrought in the 1990s. This new field has already seen the creation of several new international conferences and publication of a new journal covering the results obtained in the field.

On the other hand, in the computer science field, so-called 'machine learning' techniques, in which computers automatically extract rules from a number of given examples and which have been extensively well-researched since the early 1960s or earlier, have reached a matured age in the 1990s (Shavlik, 1998). Given the above-described circumstances in the molecular biology field, in which we have to deal with a large amount of sequence data, we can expect machine learning to be put to particularly effective use for a variety of computational problems in molecular biology (Hunter, 1993). Since the biological data which has been determined through rather manual experiments is liable to contain certain errors or noise, robust learning approaches such as stochastic rule learning will likely be especially useful as a means of overcoming these problems.

Purpose of the Thesis

In light of these considerations, we have established new knowledge representations, especially stochastic knowledge representations, for some crucial problems in molecular biology, and we have obtained valuable results from our computational experiments. In addition to these results, we have developed new machine learning algorithms which are well suited to helping solve the problems in molecular biology.

We here note that the strategies which we will propose in this thesis are strongly related to predicting protein structures, since the protein structure prediction problem is the single most important problem in the molecular biology field with its broad scientific and engineering applications. Existing approaches proposed to solve the problem have been far from satisfactory to solve for molecular biologists though a number of attempts have been made on the problem through the use of computers.

In the thesis, Chapters 3 and 6 deal with predicting protein secondary structures (α-helix and β-sheet, respectively) which are regularly and frequently seen in protein 3-dimensional structures. The α-helix prediction method based on stochastic rule learning, which is proposed in Chapter 3, achieved residue-wise prediction accuracy of over 80%, which was almost the same level as that of PHD (Profile network from HeiDelberg) (Rost, Sander, & Schneider, 1994) which had been the most powerful prediction method available. The β-sheet prediction method using stochastic tree-grammar learning, which is proposed in Chapter 6, provides a complete new approach toward representing β-sheet structures, which has not been tried yet in any secondary structure prediction method or in any 3-dimensional prediction method.

The main purpose of this thesis is to establish novel stochastic knowledge representations and machine learning strategies for the crucial problems in biological sequence analysis, and to demonstrate in terms of computational experiments that our methods including two ones described above, are effective means of addressing the problems. We believe that the methods developed in our research will contribute greatly to both the computer science and molecular biology fields.

In the following four sections, we briefly summarize the content of our four methods and the problems to which each of them is applied, i.e. predicting α-helices with stochastic rule learning, representing sequences with probabilistic networks, supervised learning of hidden Markov models for sequence discrimination, and predicting β-sheets based on stochastic tree grammars.

Predicting α-helices with Stochastic Rule Learning

α-helices and β-sheets are the two most crucial secondary structures, which are, in the biology field, defined as regular structures often seen in protein 3-dimensional structures. Predicting secondary structures for a given new sequence, in which each residue of the sequence is assigned by one of the three labels (α-helix, β-strand, or others) of secondary structures, is thought to be a crucial problem, since it can be a step toward predicting global 3-dimensional structures of proteins. This problem has been considered for a long time since the early 1970s, but even at present, no very satisfactory solution has been proposed (Cuff & Barton, 1999).

Under this circumstance, we focused on predicting only α-helix regions whose structural properties could possibly be determined by the local region, and aimed at predicting the region with high accuracy, based on the theory of stochastic rule learning.

The biggest feature of the method is that it defines novel stochastic rules for α-helix regions, and that it establishes a new strategy which optimizes the rules, clustering amino acids at each residue position of an α-helix region of given data. The strategy which uses physico-chemical properties of the amino acids is based on Rissanen's minimum description length (MDL) principle (Rissanen, 1978, 1989) derived from the information theoretic literature. Our method obtains optimal clustering for types of amino acids in the context of information theory as well as compensates for the current small amount of protein 3-dimensional structure data.

Another feature of the method is the use of sequences which have appropriate similarity to a sequence whose 3-dimensional structure is already known. This approach increases the number of available sequences to construct rules predicting α-helices, and greatly improves the prediction accuracy. This type of using sequences with unknown structures is currently popular in secondary structure prediction methods, but we first proposed the idea in 1992. At that time, no other methods had ever used this idea, to the best of our knowledge.

In our experiments, we learned stochastic rules from 25 training proteins and their homologous sequences to predict α-helix regions in test proteins, which consist of more than 5000 residues with 38% α-helix content. Each of these test sequences possesses less than 25% homology to any protein in the training sequences. Our method achieved approximately 81% average prediction accuracy for the test sequences, which compared favorably to Qian and Sejnowski's existing secondary structure prediction method (Qian & Sejnowski, 1988), which attained no more than 75% average accuracy. The 81% accuracy matched the highest level attained with Rost and Sander's method (Rost et al., 1994) which had proven to be one of the best secondary structure prediction methods.

Another important advantage of our method is that we can exhibit a comprehensible rule used for predicting an α-helix region, while both the Qian and Sejnowski's method and the Rost and Sander's method cannot. We believe that this is a striking merit in a situation in which we need to find multiple similar α-helix regions distributed over various types of sequences.

Representing Motifs with Probabilistic Networks

In general, the term motif indicates a common short pattern of multiple sequences which have a certain functional or structural (especially functional) feature (Bork & Koonin, 1996). In other words, a motif can be regarded as a key pattern which specifies the feature of the sequence in which the motif is found.

For a local region such as the motif, if the relations between the residues contained in the region can be analyzed with computers automatically, it would be a great aid to understanding the mechanism of the function related to the region. To analyze such types of inter-residue relations, we propose a new method for representing inter-residue relations of a local region in a protein sequence as a probabilistic network.

Our method produces, from a large number of sequences of a local region, a network which describes relations to be considered among amino acid residues in the region. Based on an efficient greedy-search algorithm and the minimum description length (MDL) principle (Rissanen, 1978, 1989), we establish a new algorithm which constructs a near-optimal network in the context of information theory as well as estimates probabilistic parameters of the network.

In our experiments, we constructed a probabilistic network for the EF-hand motif which is common to calcium-binding proteins. Experimental results show that our method provides a visual aid to seeing inter-residue relations of the motif using a probabilistic network, and the network captures several important structural features which are peculiar to the motif. Furthermore, we compared our method with neural network based method, in terms of the motif classification problem, and the result shows that the two methods achieved almost the same classification accuracy while neural networks do not provide any visual information on inter-residue relationships in the motif.

Supervised Learning of Hidden Markov Models

Hidden Markov models have been proposed as stochastic models for representing multiple sequences which are categorized in a class having either a common 3-dimensional structure or function (Durbin, Eddy, Krogh, & Mitchison, 1998). So far, such representation for multiple sequences has been done in a form referred to as a 'profile' (Lüthy, Xenarios, & Bucher, 1994), which corresponds to a kind of numeral distribution for twenty types of amino acids and represents a typical sequence of the class to which the multiple sequences belong. The profile is obtained by first aligning multiple sequences belonging to a class and next calculating the distribution at each position of aligned sequences.

On the other hand, the biggest advantage of the hidden Markov models is that there is a conventional learning algorithm for the models, and that a profile for multiple sequences can be automatically obtained by training the parameters of the models using the algorithm, which is generally called the Baum-Welch (Rabiner, 1989). However, since the algorithm uses only sequences belonging to a class to train a hidden Markov model representing the class, there is a problem that the trained hidden Markov model cannot provide performance enough to discriminate sequences belonging to a class from other sequences (Brown et al., 1993).

Under the circumstances, we established a new learning method for hidden Markov models to discriminate unknown sequences with high accuracy. Our method sets a function which corresponds to the error-distance between the observed output given by a hidden Markov model and the desired one, for each sequence, and using a gradient descent algorithm (Rumelhart, Hinton, & Williams, 1986), trains the parameters of the hidden Markov model so that the function should be minimized. The biggest feature of the method is that our method allows hidden Markov models to use data in which each example has its own label. For example, our method can use not only the sequences belonging to a class which should be represented by the hidden Markov model, but also the sequences which do not, i.e. negative examples. In short, our method allows us to realize the *supervised learning* of hidden Markov models. This feature also improves their prediction accuracy for unknown sequences while maintaining computational complexity on the same order as those of the conventional methods.

We evaluated our method in a series of experiments, and compared the results with those of two existing learning methods for hidden Markov models, including the Baum-Welch algorithm, and a neural network learning method. Experimental results show that our method makes fewer discrimination errors than the other methods. From the results obtained, we conclude that our method which can allow us to use negative examples is useful for training hidden Markov models in discriminating unknown sequences.

Predicting β-sheets Based on Stochastic Tree Grammars

As mentioned earlier, predicting protein secondary structures have stayed at an unsatisfactory level for nearly 30 years, though the problem has been studied extensively. The main reason for this disappointing result lies in the difficulty in predicting β-sheet regions, because there are unbounded dependencies exhibited by sequences containing β-sheets.

To cope with this difficulty, we defined a new family of stochastic tree grammars, which we call stochastic ranked node rewriting grammars (SRNRGs) (Mamitsuka & Abe, 1994a), which are powerful enough to capture the type of unbounded dependencies seen in the sequences containing β-sheets, such as the 'parallel' and 'anti-parallel' dependencies and their combinations. The learning algorithm we established is an extension of the conventional Baum-Welch algorithm for hidden Markov model learning, but with a number of significant modifications.

First, we restrict the grammars to represent β-sheets corresponding to the subclass of SRNRG, and devise simpler and faster algorithms for this subclass. Secondly, we reduce the number of amino acids by optimally clustering them using their physico-chemical properties and the minimum description length principle, gradually through the iterations of the learning algorithm. This modification contributes to improve prediction accuracy for unknown sequences as well as to make up for the small size of sequences with known structures. Finally, we parallelize our prediction algorithm to run on a highly parallel computer, a 32-processor CM-5, and are able to deal with a long sequence having approximately 250 residues in a practical time, which sequence usually requires several weeks to be predicted on a single processor machine.

We applied our method to real protein data as a tool for scientific discovery of common β-sheet structures in proteins whose sequences are dissimilar to one another. In particular, using our prediction method based on stochastic tree grammars, we attempted to predict the structure of β-sheet regions (of reasonable complexity) in a large number of arbitrarily chosen proteins, using only training sequences from dissimilar proteins. The experimental results indicate that the prediction made by our method on the approximate locations and structures of 'two sequentially adjacent hairpin β-strand motifs' (which coincide with the class of four-strand β-sheets representable by 'rank 1' SRNRG) is statistically significant.

Our method was able to actually predict a couple of four strand β-sheet structures approximately correctly, based on stochastic grammar fragments trained on sequences of proteins that are very different in sequence similarity from that of the test sequence, thus discovering a hitherto unnoticed similarity that exists between local structures of apparently unrelated proteins. Also, in the course of our experiments, it was observed that the prediction is much easier when we restrict the test sequences to contain relatively isolated β-sheets only, and exclude partial β-sheets existing as part of a larger β-sheet structure. Surprisingly, it was found that for prediction of these partial β-sheet structures, training data from relatively isolated β-sheets were not only useless but even harmful. These observations together suggest that: (i) There exist some similarities between the sequences of relatively isolated four strand patterns in different proteins, and acquiring generalized patterns for them can help improve prediction accuracy. (ii) Satisfactory prediction of larger β-sheet structures would probably require more global information than the level of four strand patterns.

Organization of the Thesis

The following is the organization of the chapters that follow:

Chapter 2 gives a review of molecular biology and machine learning. The review of molecular biology starts from basic knowledge of the field and then describes three types of problems in molecular biology that we will attempt to solve through the use of computers. All three problems correspond to the work described in Chapters 3 to 6, and the relation between each chapter and the problem(s) which it deals with is shown in Figure 1.2. Note that the hidden Markov models dealt with in Chapter

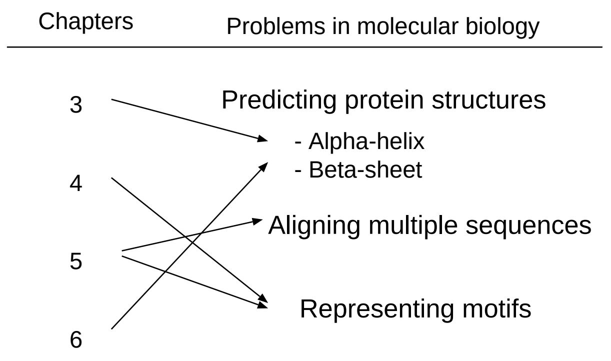

Chapters
Problems in molecular biology
3
4
5
6
Predicting protein structures
- Alpha-helix
- Beta-sheet
Aligning multiple sequences
Representing motifs

Figure 1.2: Relations between Chapters 3 to 6 and problems in molecular biology

5 have proven to be effective for multiple-sequence alignment and motif representation, and thus two arrows are drawn from Chapter 5. The review of machine learning also starts from basic knowledge in the field and describes four classes of models and three types of learning algorithms. The four models are repeatedly defined in Chapters 3 to 6 in the order they were introduced, and the relation between each model and the learning algorithm which is proposed to it is shown in Figure 1.3. Note here

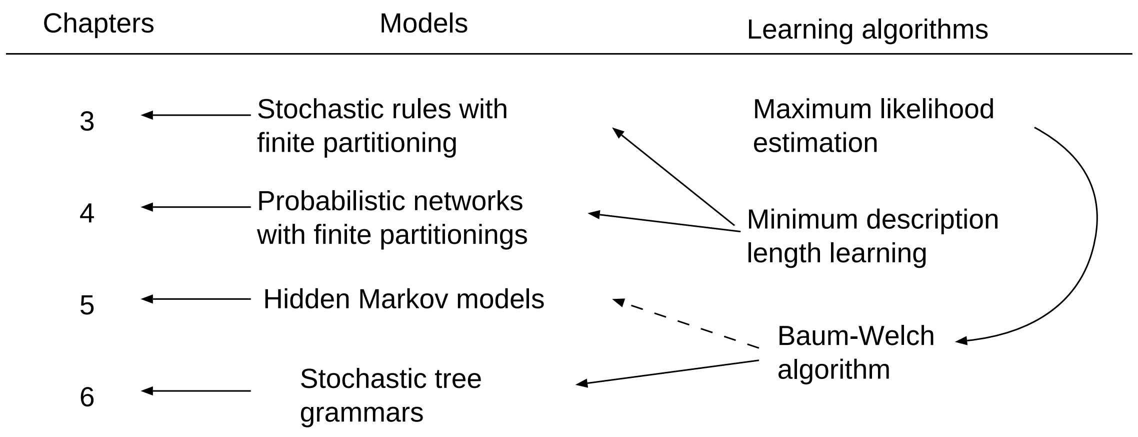

Chapters
Models
Learning algorithms
3
Stochastic rules with finite partitioning
4
Probabilistic networks with finite partitionings
5
Hidden Markov models
6
Stochastic tree grammars
Maximum likelihood estimation
Minimum description length learning
Baum-Welch algorithm

Figure 1.3: Relations between Chapters 3 to 6 and machine learning strategies

that maximum-likelihood estimation is not used in our work to train any of the four types of models directly, but the Baum-Welch algorithm is based on the idea of the maximum likelihood. Further note that the Baum-Welch algorithm is a general learning method for hidden Markov models but that our algorithm proposed in Chapter 5 is better in terms of experimental results conducted using actual sequences, and so in Figure 1.3, we connect the hidden Markov models and the Baum-Welch algorithm with a dotted arrow.

Chapter 3 introduces a new method for the problem of predicting α-helices in a given new sequence, based on stochastic rule learning. First, we discuss the necessity of stochastic rule learning for the problem, and briefly review the history of the approaches which have been taken to the problem of predicting α-helices. Next, we define our stochastic rules and describe in detail our new strategy for learning stochastic rules, based on Rissanen's minimum description length (MDL) principle and the physico-chemical properties of amino acids, and to predicting α-helix regions in a given sequence using the trained stochastic rules. Finally, on which the model was trained, we present favorable experimental results, comparing the method with existing methods, and discuss subjects to be tackled in the secondary structure prediction problem.

Chapter 4 introduces a probabilistic network to represent inter-residue relations hidden in given amino acid sequences and our new method to train that network from the given sequences. First, we mention several current approaches used to represent motifs – such as string patterns or profiles – and the necessity of considering inter-residue relations in representing them. Next, we describe in detail our new efficient method of constructing a probabilistic network automatically from given examples, based on the MDL principle and greedy-search. Finally, we show the results obtained by applying our method to EF-hand motif sequences.

Chapter 5 introduces a new supervised learning algorithm to train hidden Markov models. First, we mention a variety of current applications of hidden Markov models in the field of computational molecular biology and the problems encountered. Next, we describe in detail our new method for learning a hidden Markov model, training the system from not only examples which belong to the class to be represented by the model, but also examples which do not. Finally, we present favorable experimental results, comparing with two other existing methods, including the Baum-Welch algorithm, which is currently the most popular algorithm to train hidden Markov models.

Chapter 6 introduces a novel strategy for learning stochastic tree grammars to capture the long-range interactions which give rise to β-sheets, and to predict both the locations and structures of β-sheets in a given new sequence. In this chapter, we define stochastic tree grammars and describe our newly devised method for training them from given examples and using them to predict β-sheets in a given new sequence with the trained grammars. Finally, we show experimental results obtained by applying our method to a protein database, and describe possible future work to improve our predictive performance.

Chapter 7 gives concluding remarks of this thesis, and discusses the models and learning algorithms which should be considered in improving the accuracy of our current models and learning methods.

Chapter 2

Problems in Molecular Biology and Machine Learning

2.1 Computational Problems in Molecular Biology

In this section, we first present basic knowledge in molecular biology and next describe three computational problems in the field – protein structure prediction, multiple sequence alignment and motif detection – which comprise the subjects of this thesis, as well as being the principal themes of current computational molecular biology.

The first problem, which is that of predicting protein structures for a given amino acid sequence through the use of computers, is the most crucial and difficult problem in the field and is the central theme of this thesis. The problem of predicting secondary structures, in particular, has been under study since the 1970s, but satisfactory results have not yet been reached. The methods of predicting α-helices and β-sheets, which are the major two secondary structure classes, are discussed in Chapters 3 and 6, respectively.

The second problem is that of aligning multiple sequences and using the result of the alignment, and of building a sequence 'profile' which corresponds to a representation form for all the aligned sequences. Sequence alignment is a very basic problem, and has been a goal in computational biology since the early 1970s. Aligning multiple sequences belonging to a class has already been achieved on a practical level, at which the profile for the class, which is obtained by the alignment, is used as a tool for database search. In recent few years, however, a statistical representation newly applied in this field – 'hidden Markov model' – has been proposed for use both in aligning sequences and in calculating profiles of the sequences at the same time. Chapter 5 is concerned with this problem, and describing a new learning algorithm for hidden Markov models.

The third problem is related to 'motifs', which typically indicate common patterns of biological sequences belonging to a certain class. The main research theme in this problem is the best method to use to represent a motif in order to detect the motif with high accuracy in given sequences. Motifs which had been simply represented by string patterns so far, are also being replaced by new statistical representations such as hidden Markov models. Chapters 4 and 5 deal with this problem and Chapter 4 proposes a new method for representing motifs, instead of simple string patterns.

The above three problems are not the only problems in molecular biology. In addition, well-known major computational problems include finding genes in nucleotide sequences, phylogenetic tree reconstruction, predicting the structure of nucleotide sequences, etc. These problems remain unsolved, despite being well-researched in the computational molecular biology field.

2.1.1 Fundamentals of Molecular Biology

Basic Terms

We will first explain the relation between several basic terms such as nucleotides, amino acids and proteins. Genetic information is stored as a sequence of four types of *nucleotides*, i.e. adenine (A), thymine (T), guanine (G) and cytocine (C), which are the basic units of genes. Each triple of nucleotides in the sequence unambiguously specifies one of twenty types of *amino acids*, and linked amino acids, which are translated from a nucleotide sequence in the order of the triplets (or *codons*) are bound to each other to form a *protein*. Figure 2.1 shows this basic flow from genetic information

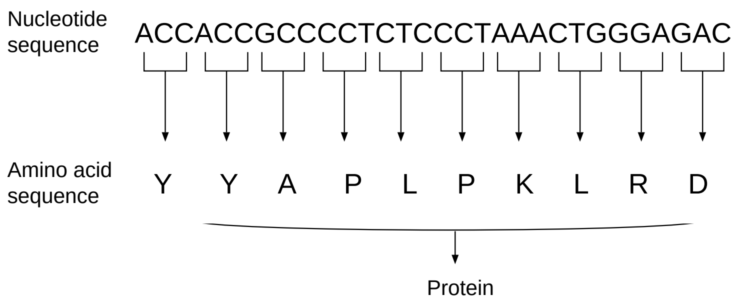


Figure 2.1: Basic terms in molecular biology

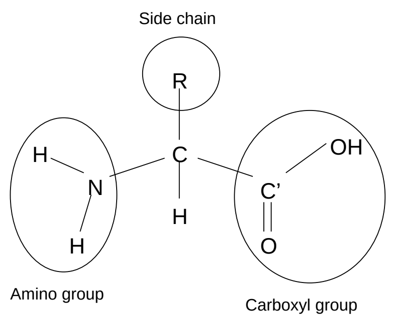
to protein.

In other words, a protein is a sequence of amino acids – the basic unit of proteins – each of which is a small molecule consisting of roughly ten to thirty atoms. Figure 2.2 shows the molecular structures of an amino acid and an amino acid sequence, where R is called the *side chain* which identifies the type of the specific amino acid. In Figure 2.2 (b), the portion except the R is called the *backbone*. Figure 2.3 shows twenty types of amino acids specified by the side chains. As the types of amino acids used in proteins is limited to the same twenty in any living organisms, a protein can be identified by a sequence of twenty different characters. The term *residue* is also used to indicate an amino acid, and a relatively short sequence of the amino acids is called a *peptide*. Note that in this thesis, we do not deal with nucleotide sequences but rather amino acid sequences only.

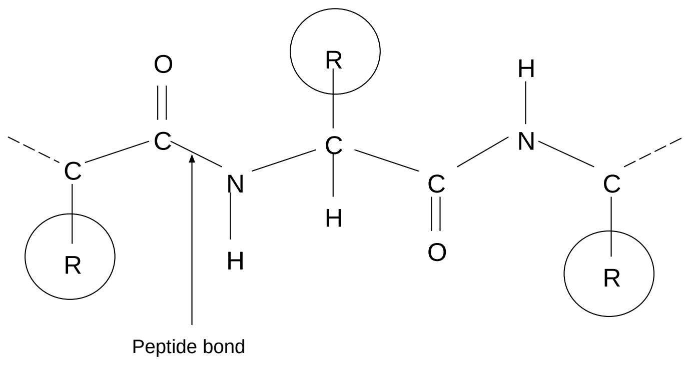
As mentioned earlier, there are three major databases of nucleotide sequences in the world. Similarly, there are two major databases for amino acid sequences, that is, the PIR (Protein Identification Resource) (W. C. Barker et al., 1999) database in the USA and the Swiss-Prot (Bairoch & Apweiler, 1999) database developed by Bairoch in Europe. As of January 1999, Swiss-Prot Release 37.0 contains 79626 sequence entries and over twenty-eight million amino acids.

Hierarchy of Protein Structures

The amino acid sequence is referred to as the *primary structure* of a protein. Since each amino acid is a molecule with its own 3-dimensional structure, any protein, corresponding to a sequence of the amino acids, also has its own 3-dimensional structure. Partial structures which are regularly and frequently seen in the proteins' 3-dimensional structures are referred to as *secondary structures*. The *tertiary structure* of a protein indicates the packing of structural elements including the secondary structures, and a collection of the tertiary structures is referred to as a *quaternary structure*. Figure 2.4 shows an example of this hierarchy of protein structures.



(*a*)



(*b*)

Figure 2.2: Amino acid and amino acid sequence

Typical and well-known secondary structures are *α-helices* and *β-sheets*.

The α-helix is a literally helical structure held in form by hydrogen bonds within the structure and exists as a single connected local region of an amino acid sequence. The length of an α-helix is in the range of approximately five to forty residues, with the average being ten to fifteen residues. Figure 2.5 (a) shows a 3-dimensional view of the α-helix.

In contrast, a β-sheet is made up of multiple local regions, two to ten residues in length, called β-strands. These are collected together next to one another side by side, either in parallel or anti-parallel, and there are hydrogen bonds between adjacent β-strands. Notationally, parallel and anti-parallel β-sheets of four β-strands (each with two residues (letters), a and b) are denoted as *..ab..ab..ab..ab..* and *..ab..ba..ab..ba..*, respectively, to represent their primary structures, where the use of an identical letter indicates that those positions face each other in β-sheet structure (and are believed to be correlated). Further note that a β-sheet is composed of multiple mutually distant local regions in an amino acid sequence, which are distributed over wide areas of the sequence and are bound to each other via long-range interactions by hydrogen bonds. Figure 2.5 (b) shows the 3-dimensional view of the β-sheet, and Figure 2.6 shows the molecular structures of the two types of β-sheets.

There is a database for protein 3-dimensional structures, which is referred to as the Protein Data Bank (typically abbreviated to 'PDB') (Bernstein et al., 1977). As of April 1999, the PDB contains 9631 protein structure coordinate entries.

Protein Family

Proteins which have a similar function or a chemical behavior are categorized as forming a protein *family* which corresponds to a sub-category of a protein *superfamily*. The superfamilies are typically regarded as categories of classification relative to protein function. The biggest and most typical well-known protein superfamily is the globin superfamily, which accounts for over a thousand of sequences in the PIR database, including Hemoglobins, Myoglobins, Hemoproteins, etc. In the PIR, all sequences are listed in the order of the family and superfamily.

Similarity / Identity / Homology

The terms *similarity* and *identity* are used synonymously in this thesis, and both typically indicate how many residues of two sequences coincide when aligned. For example, when we say that a sequence has

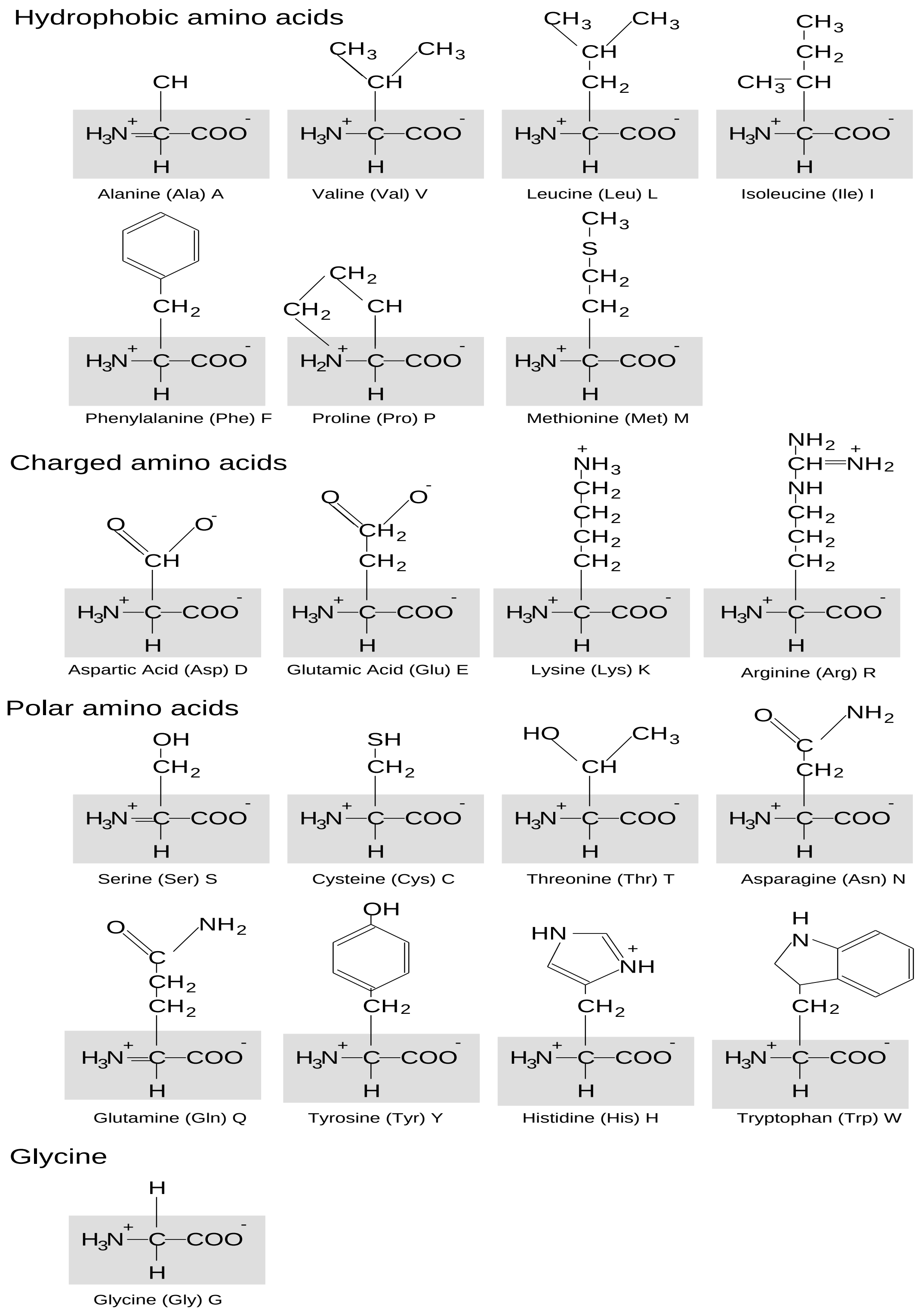


Figure 2.3: 20 types of amino acids

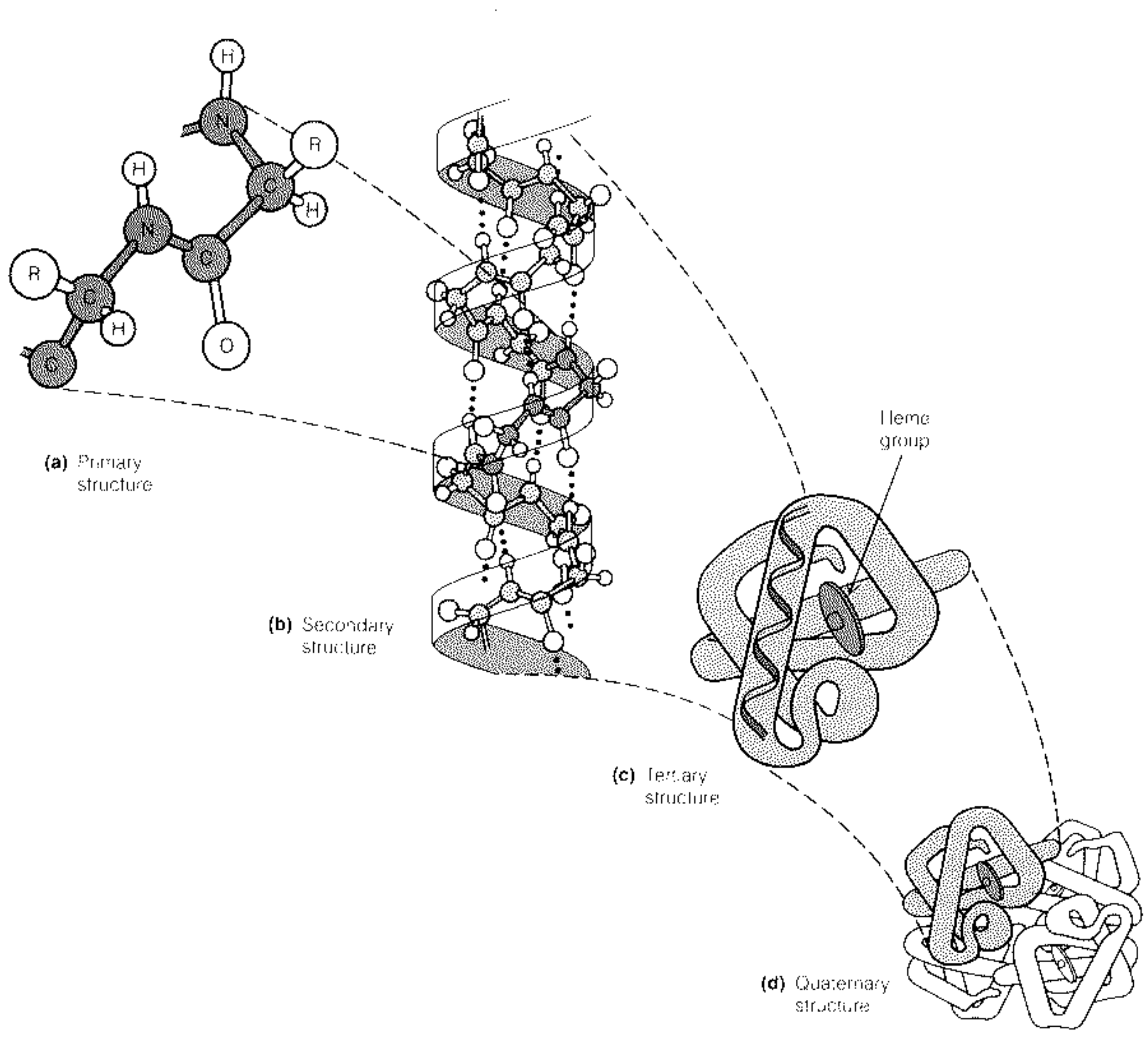
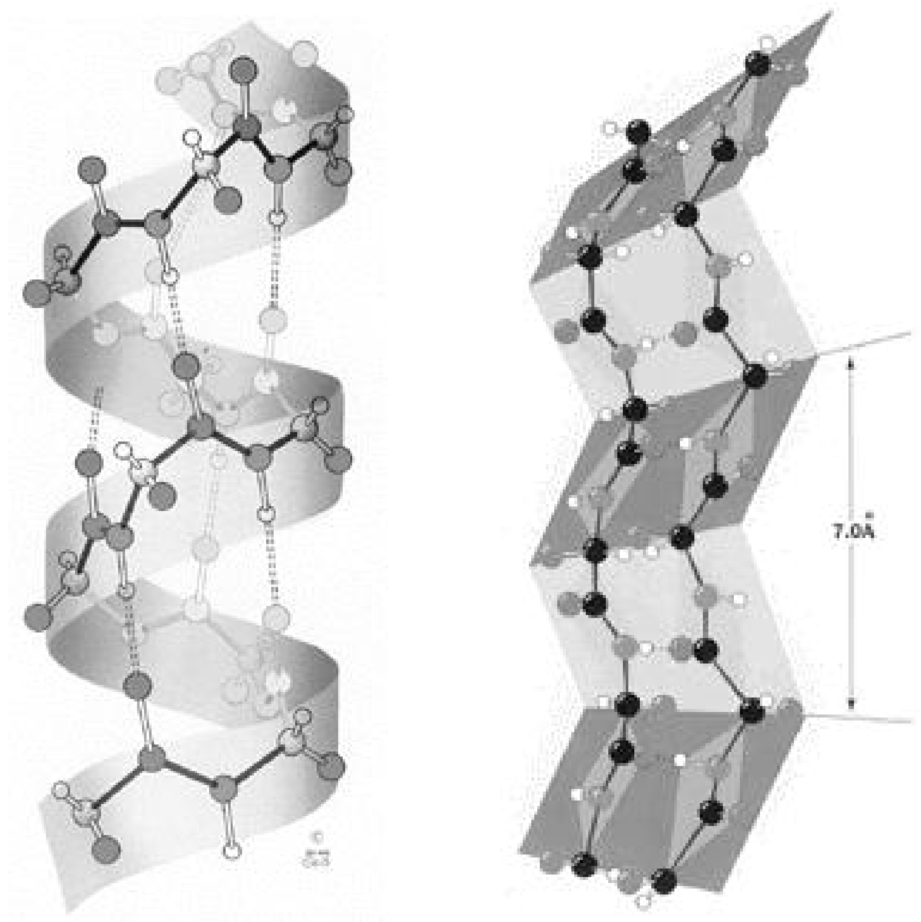


Figure 2.4: Hierarchy of protein structures

from http://www.eng.rpi.edu/dept/chem-eng/Biotech-Environ/PRECIP/precpintro.html



(*a*) (*b*)

Figure 2.5: Secondary structures : (a) α-helix (b) β-sheet
from (Voet & Voet, 1990)

25% sequence identity to another sequence, 25% of the residues of the sequence match the residues in the other one when aligned. The term *homology* is also synonymous with them, but homology includes a nuance of structural identity. Thus, when two sequences share at least 25% sequence identity and each is at least 80 residues in length, then the two sequences are said to be *homologous* because they certainly hold a common 3-dimensional structure (Doolittle, 1986; Sander & Schneider, 1991). Furthermore, the term homology contains more of an evolutionary nuance, i.e. that two sequences coincide evolutionarity. Thus, in this thesis, we use the terms similarity and identity when the evolutionary relation of two sequences is unknown.

Folds / Folding

When a protein is synthesized from its nucleotide sequence in a living organism, the process by which the protein's tertiary or quaternary structure is formed from the protein's secondary or primary structure is referred to as *folding*. A pattern of protein structure produced by folding is referred to as a *fold*. That is, the fold also indicates a class of protein 3-dimensional structure, and this is different

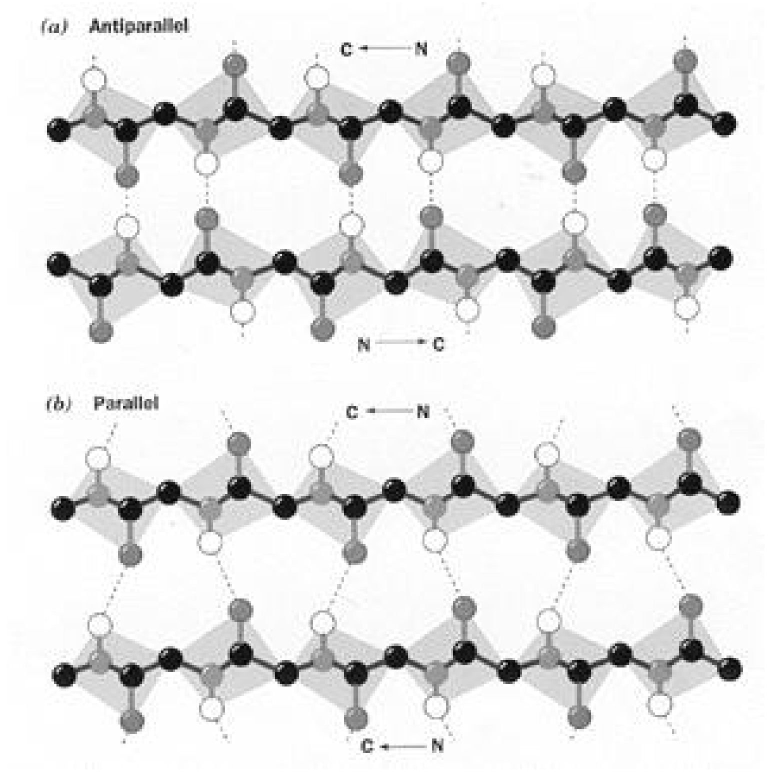


Figure 2.6: Antiparallel and parallel β-sheets
from (Voet & Voet, 1990)

from the superfamily, which is classified in terms of protein functions.

2.1.2 Protein Structure Prediction

Prediction of 3-Dimensional Protein Structure

The 3-dimensional structure of a protein is closely related to the protein's specific function, because possible functions are restricted by structure. Furthermore, understanding the 3-dimensional structure of a protein is the first goal of the molecular biologist who determines its sequence and who hopes to elucidate biological phenomena involving the protein. Accordingly, in the field of molecular biology, it is crucial to determine the 3-dimensional structure of protein.

To date, 3-dimensional protein structures have been determined only via experimental techniques such as X-ray crystallography (Brünger & Nilges, 1993) or nuclear magnetic resonance (NMR) (Wüthrich, 1989). However, these approaches require experience-trained experts or knowledge of special techniques and, consequently are expensive in terms of both money and time for most molecular biologists.

For this reason, to efficiently predict the 3-dimensional structures of a given amino acid sequence through the use of computers is the most crucial and extensively considered current problem in the field. We can roughly classify the methods of predicting 3-dimensional protein structures with computers into two categories: energy minimization based on physical chemistry, and all others.

The energy minimization approach includes two sub-categories : molecular orbital calculation and molecular dynamics (Boczko & Brooks III, 1995). These two approaches (especially the former)

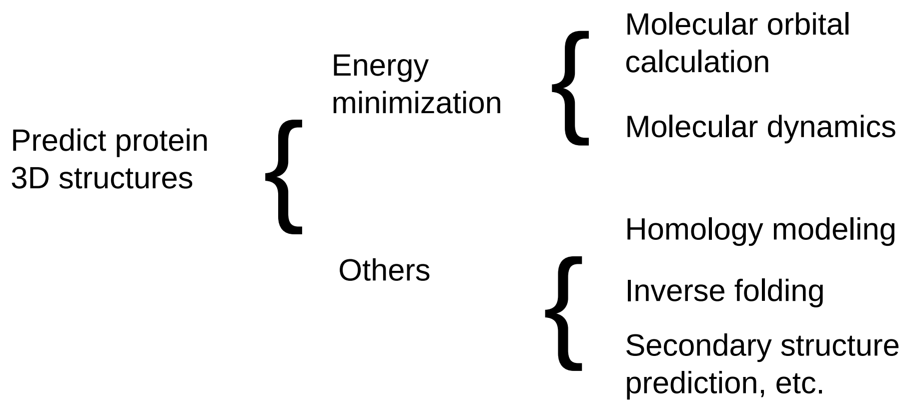

Predict protein 3D structures
Energy minimization
Molecular orbital calculation
Molecular dynamics
Others
Homology modeling
Inverse folding
Secondary structure prediction, etc.

Figure 2.7: Classification of the methods for predicting protein 3-dimensional structures

require a large amount of computation time and memory space for dealing with the folding of an entire protein, starting with the protein in a random structure, and hence the biggest problem of these methods is that the resources they require surpass practical limits even with the highest performance computers currently available. Thus, they cannot be considered as efficient approaches, and hereafter, so we will not mention them further in this thesis.

Numerous other approaches have been proposed for predicting proteins' 3-dimensional structures through the use of computers as well. For the most part, these can be roughly classified into three categories (Moult et al., 1997; Shortle, 1995; Hubbard et al., 1996): (i) *homology modeling* approaches (c.f. Sánchez & Sali, 1997), (ii) *inverse folding* approaches (c.f. Jones, 1997) – also known as *remote homology modeling* or *threading* – and (iii) others. A summary of this classification of methods of predicting protein structures is shown in Figure 2.7.

These prediction approaches basically use only the information in the input amino acid sequence, namely the sequence of twenty residue-code characters, and hence they are based on the Anfinsen's hypothesis (Anfinsen, 1973) that the 3-dimensional structure of a given protein is determined by its amino acid sequence alone. Furthermore, each of these approaches has its own merits and demerits, but what is common between them is that they are all far from achieving the level at which the 3-dimensional structure of an arbitrary amino acid sequence can be predicted with reasonable accuracy and confidence.

The idea behind homology modeling is based on the general tendency of homologous proteins to have common 3-dimensional structures. More concretely, new sequences having over 25% sequence similarity to other sequences with known 3-dimensional structures have proven to be most likely to fold into the same 3-dimensional structure (Sander & Schneider, 1991). Thus, in homology modeling, when a new sequence is given, a protein having a similar sequence to it is searched for in existing protein-structure databases such as PDB, as shown in Figure 2.8. Homology modeling approaches have been used since the late 1980s, around which time the amount of protein 3-dimensional structure data available started to increase rapidly.

Inverse folding approaches are also based on sequence homology, but endeavor to enhance the predictive ability of homology modeling by supplementing it with other prior knowledge. The inverse folding method effectively keeps a catalogue of patterns of amino acid sequences corresponding to existing types of protein structures, and predicts the structures of new sequences by simply finding the most appropriate pattern that matches the input sequence.

In confirmation of such homology modeling or inverse folding approaches, early in the 1990s, Chothia (Chothia, 1992) put forward the controversial proposal that all 3-dimensional protein structures (foldings) found in today's living organisms can be classified into a relatively small number (less than a thousand or so) of categories.

However, these two approaches (especially homology modeling) require, for any meaningful prediction to be possible, that the new input sequence be reasonably similar (over 25%) to some sequence with known structure already in a database (Sander & Schneider, 1991), and consequently, the empirical results obtained so far have been limited in their scope (O'Donoghue & Rost, 1995).

Most approaches in the third category, on the other hand, begin by predicting the secondary structure of a protein, before attempting to predict its entire 3-dimensional structure. Secondary structure prediction methods have been tried over decades as will be mentioned but results have remained at an unsatisfactory level of accuracy, and the information that is provided by such classical methods is not really sufficient to serve as a basis of 3-dimensional structure prediction.

The method proposed in Chapter 6 of this thesis for predicting β-sheet structures is deeply related to these three types of 3-dimensional protein structure prediction methods. Our method focuses on β-sheet structures, which can be regarded as the frameworks of proteins' 3-dimensional structures, and captures β-sheet structures' long-range relations as a tree grammar. Thus, the method is basically a secondary structure prediction method, but can also predict the rough 3-dimensional structure of a protein. More interestingly, we believe that our method can predict such rough 3-dimensional structures with higher accuracy than existing homology modeling or inverse folding methods, because the sequences in the β-sheet regions dealt with in our method are generally rather well conserved and so are easily predicted.

Figure 2.8 shows the flow of the process of predicting the 3-dimensional structure of a new protein (the target sequence in the figure). As shown, besides homology modeling, inverse folding and secondary structure prediction, multiple sequences alignment and motif detection (the "PROSITE" in the figure is a motif database), both of which will be described later, are also used, and are crucial in predicting protein structures.

Prediction of Protein Secondary Structure

Predicting the secondary structure for a given sequence is the classical and most well-researched problem in computational molecular biology. This problem has dealt with the three-state prediction problem, which is defined as the assignment of one of the three labels (α-helix, β-strand, or others) to indicate the secondary structure at each of the amino acids in the input sequence.

Numerous approaches have been proposed for this problem since the late 1960s and early 1970s. Here we review the history of this prediction, focusing on several major methods.

The most well-known approaches are Chou-Fasman's method (Chou & Fasman, 1974a, 1974b), Lim's method (Lim, 1974) and Garnier-Osguthorpe-Robson's method (Garnier et al., 1978). All of these methods tried to represent a single general relation between each secondary structure and a local regular pattern of amino acids within a sequence, but their average prediction accuracies were reported to be less than 60%.

In the 1980s, different types of approaches (e.g. Cohen et al., 1986; Lathrop et al., 1987) were proposed, the ideas of which were close to that of homology modeling. They first choose, from a databank, a partial protein sequence which is identical to the part of the input sequence, and then repeat the process with other parts of the input sequence, one after another. Finally, they predict the secondary structures in the input based on the multiple partial sequences obtained. Prediction accuracy was successfully improved by a small amount using these approaches, but none of them

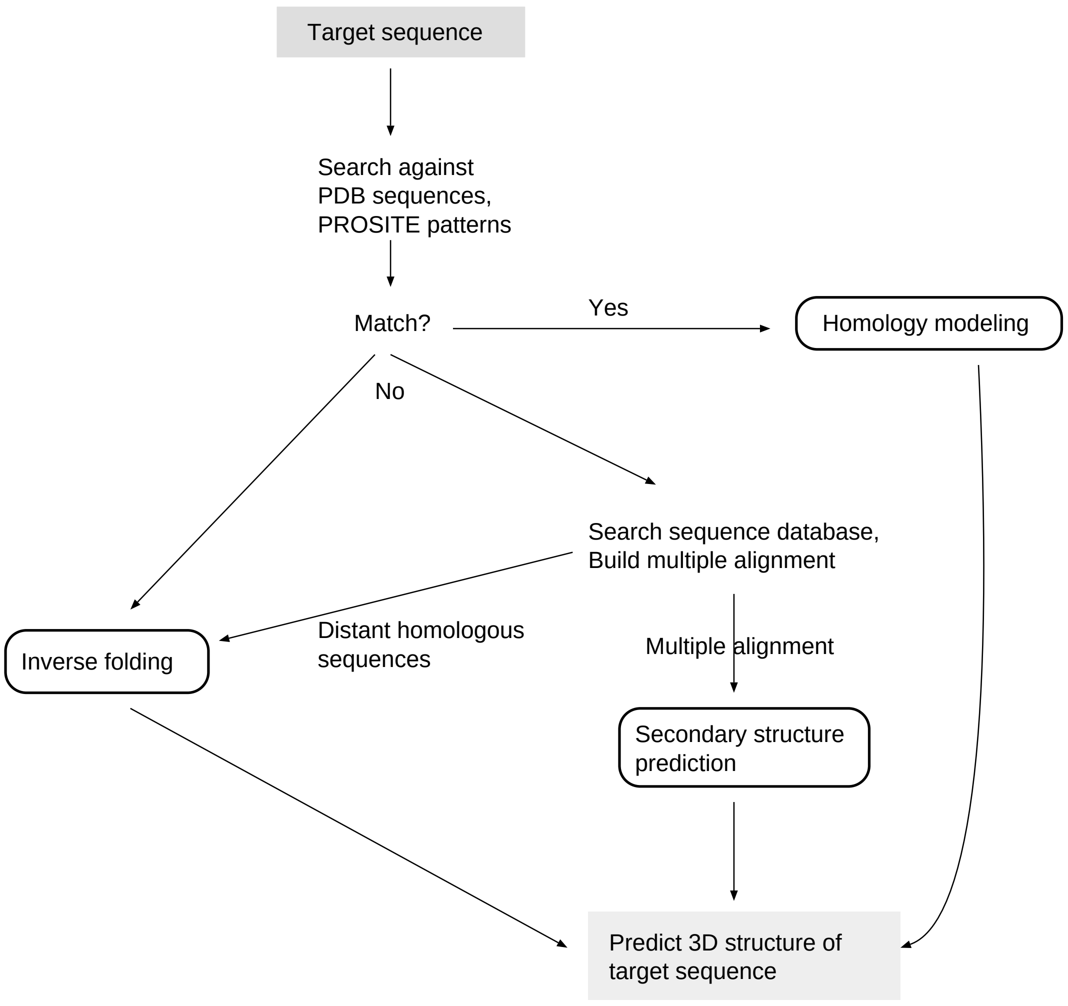

Target sequence
Search against PDB sequences, PROSITE patterns
Match?
Yes
Homology modeling
No
Search sequence database, Build multiple alignment
Inverse folding
Distant homologous sequences
Multiple alignment
Secondary structure prediction
Predict 3D structure of target sequence

Figure 2.8: Flow of the process of predicting protein 3-dimensional structures

achieved average accuracies much greater than 60% for the three-state prediction problem.

In the late 1980s, machine learning techniques such as neural network learning (Qian & Sejnowski, 1988; Holley & Karplus, 1989) or inductive logic learning (King & Sternberg, 1990) started to be applied to this problem. In particular, the neural-network based approach improved the average prediction accuracy for the three-state prediction problem to more than 65%.

From 1990 to 1993, a number of groups (Kneller et al., 1990; Zhang et al., 1992; Stolorz et al., 1992; Hayward & Collins, 1992) proposed various modified versions of neural-network based methods. In particular, PHD (Profile network from HeiDelberg), proposed by Rost and Sander (Rost & Sander, 1993a), solved to some degree one of the biggest problem in secondary structure prediction, i.e. the lack of proteins whose 3-dimensional structure is already known, by aligning multiple sequences whose 3-dimensional structures' were unknown to the sequence whose 3-dimensional structure were known. By doing this, they improved average accuracy until finally approaching 70% for the three-state prediction problem. In fact, PHD is the most effective approach in the midst of 1990s for solving the secondary structure prediction problem.

Against this background, we established a method based on stochastic rule learning in Chapter 3 of this thesis, focusing on predicting α-helices only, and achieves a prediction accuracy exceeding 80% (almost the same level as that of the PHD in predicting α-helices only). Moreover, our method has the advantage of being able to provide understandable rules for predicting α-helices in a given new sequence, while the PHD cannot.

A major criticism of secondary structure prediction is that the information provided by them is essentially 1-dimensional in nature (O'Donoghue & Rost, 1995) and thus is not ideal as a basis of 3-dimensional structure prediction, even if it provides high prediction accuracy. This criticism is especially true of β-sheet prediction, since as mentioned earlier, β-sheets consist of several separated portions of a sequence while existing secondary structure prediction methods, including PHD, provide the information only as a 1-dimensional label sequence, in which a label (α-helix, β-sheet or others) is assigned to each of the amino acids in the input sequence, and do not consider the relations between mutually distant portions in the sequence, such as which β-strands belong to the same β-sheet, etc.

In response to this criticism, we established a novel method in Chapter 6, which learns not only local properties as a 1-dimensional sequence, but also the long-range relations of β-sheets. In this sense, our method is the first to break past the existing secondary structure framework in which 1-dimensional prediction is done, to extend it into a proper preliminary tool for the prediction of the global 3-dimensional structures of proteins.

2.1.3 Multiple Sequence Alignment - Profile Calculation

As shown in Figure 2.8, in the homology modeling approach to prediction of protein structure, we need to search for sequences having similarity to the input sequence. To measure the similarity between two sequences, techniques for aligning sequences with reasonable accuracy and speed are required.

Any molecular biologist who determines a nucleotide sequence first tries to find similar sequences in the major databases, in which sequences whose structures and/or functions are already known are available for search. If one is fortunate enough to find a large number of similar sequences, one can start the next step of inferring the function or structure of the new sequence using them (Altschul et al., 1994).

It should be clear from the above that it is a crucial problem in molecular biology to align sequences accurately and rapidly. One major problem in aligning two given sequences (referred to as *pairwise alignment*) is that blanks (referred to as *gaps*) made up of sub-sequences to be ignored are allowed to be inserted in each of the two sequences. A method based on the dynamic programming (hereafter referred to as the DP) has been used to align sequences including gaps.

Here we briefly review the idea of the DP-based alignment, which was developed in the 1970s (Needleman & Wunsch, 1970). Figure 2.9 shows a schematic view of the alignment of two simple sequences, i.e. ACDF and ADEF. We first set up a lattice consisting of nodes and arrows, in which an arrow indicates a possible transition between states (nodes). Here, possible transitions, each of which corresponds to either the matching of two strings in the two sequences or a gap, are restricted to only three ways, i.e. left to right, top to bottom and upper left to lower right, as shown in Figure 2.9. Here, aligning two sequences with gaps can be replaced with the problem of finding the shortest path in such a lattice. Then, we number the states in the lattice from left to right (rows) and from top to bottom (columns), respectively, and let $s(i, j)$ be the *alignment score* for the state in the i-th row and the j-th column to measure the distance of the "shortest" transition to that state. Note that each transition is assigned a "length". Here, let n_r be the number of rows and n_c be the number of columns in the lattice. We can obtain $s(i, j)$ for all i, j by calculating the following equation in a nested loop

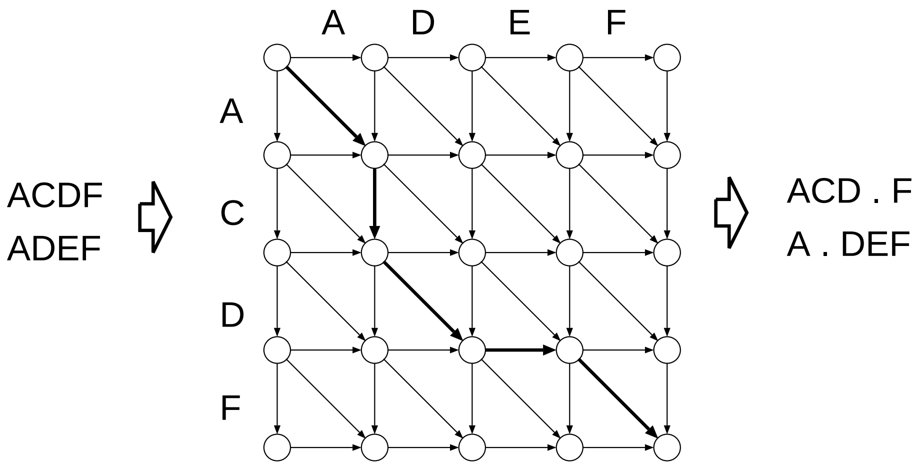


Figure 2.9: Aligning two sequences based on dynamic programming

with increasing values of i and j $(i = 1, \cdots, n_r, j = 1, \cdots, n_c)$.

$$s(i,j) = \min \begin{cases} s(i-1,j) & +Ca \\ s(i,j-1) & +Ca \\ s(i-1,j-1) & +Cb, \end{cases}$$

where Ca is a kind of *gap penalty*, Cb is the score corresponding to a direct match between a pair of a certain kind of amino acids, and $C_a > C_b > 0$. Each $s(i,j)$ which is used in calculating the final $s(i,j)$ at which $i = n_r$ and $j = n_c$, is then regarded as a state in the shortest path.

The values Ca and Cb can be varied depending on the types of the two amino acids, using a transition matrix such as the one determined by Dayhoff (Dayhoff et al., 1978). In the examples shown in Figure 2.9, the sequences are aligned as ACD.F and A.DEF. This DP-based alignment for two sequences can be expanded into one for aligning multiple sequences in several ways, such as iteration (Barton & Sternberg, 1987; Taylor, 1987; Berger & Munson, 1991) or the use of parallel computers (Ishikawa et al., 1993), but such multiple-sequence alignment has not yet been solved completely and is an active research area studied at the present time.

A *profile* is a matrix representing the proportional distribution of amino acids at each position when multiple sequences are aligned (Taylor, 1986b; Gribskov et al., 1987). For example, if we align three simple sequences, ACEF, ADEF and AGEF, the amino acids at the second position in the aligned sequences will be evenly split between C, D and G while other positions will be fixed to only one type of amino acid, so the profile for this simple example is as follows:

	1	2	3	4
A	1.0	0.0	0.0	0.0
C	0.0	0.333	0.0	0.0
D	0.0	0.333	0.0	0.0
E	0.0	0.0	1.0	0.0
F	0.0	0.0	0.0	1.0
G	0.0	0.333	0.0	0.0

A profile calculated for multiple sequences belonging to a single category, such as a family or a superfamily, is expected to represent a typical sequence or a representative model of the category.

Thus, if we have calculated the profiles for all categories of sequences, when given a new sequence, we can easily find the closest-matching category by comparing the sequence with each calculated profile or with parts of profiles. For that reason, in place of such simple amino acid distributions as that shown above, methods of representing profiles with reasonably high accuracy for specific purposes such as sequence classification have recently been one of the hottest areas of research in computational molecular biology.

In particular, in 1993 to 1994, hidden Markov models have been proposed, both for multiple-sequence alignment and calculating profiles at the same time (Krogh et al., 1994a; Baldi et al., 1994). The hidden Markov model is a powerful statistical model which allows us to handle sequences including repetition and gaps, and is at present the most influential method in multiple sequence alignment and related areas (Taubes, 1996). For example, even in the inverse folding approach to predicting proteins' 3-dimensional structures, hidden Markov models have been used as basic catalogue entries for modeling each protein structure class (Moult et al., 1997; Shortle, 1995). Details of the hidden Markov model will be described in Sections 2.2.2 and 2.2.3, and we will propose a new learning method for applying hidden Markov models to sequence discrimination in Chapter 5.

2.1.4 Motif Detection / Discrimination / Classification

The term *motif* is used in two different meanings, as occasion demands.

The first definition, which is the most general one, is as a common short sub-string pattern seen in multiple sequences belonging to a category (such as a protein family) where the region contained in the pattern corresponds to a key functional or structural (especially functional) portion of proteins in the category (Sternberg, 1991). For example, according to Release 15.0 of the PROSITE database (Hofmann, Bucher, Falquet, & Bairoch, 1999), which was established by Bairoch and is a library of this type of motifs, the sequence of any DNA polymerase B contains the motif '[YA] - [GLIVMSTAC] - D - T - D - [SG] - [LIVMFTC] - X - [LIVMSTAC]', where any amino acid from among those enclosed in [ ] is permitted at that position and any amino acid at all is permitted at the 'X', at a region supposed to be involved in binding to a magnesium ion. Motifs noted in the PROSITE database are typically five to twenty residues in length. This first definition of *motif* is synonymous with the term *consensus sequence*, which is often used to describe nucleotide sequences.

On the other hand, in the context of protein structures, *motif* can be used to represent a combination of several secondary structure elements with a specific geometric arrangement, that is frequently seen in protein structures (Branden & Tooze, 1991). The EF-hand motif and the helix-loop-helix motif are typical examples of this second definition. Most sequences classified as motifs by the first definition are not motifs by the second definition, since they are not necessarily included in secondary structure elements, because they are regions that are merely functionally crucial.

The first definition indicates that motifs should be key patterns for finding functionally crucial region of a protein. Motifs, which are currently represented by simple string patterns as in the PROSITE database, are obtained by first aligning multiple sequences belonging to a relevant class such as a superfamily, next choosing an appropriate common pattern in the aligned sequences manually, and finally collecting the extracted motifs into a database. When a molecular biologist determines a new sequence, he or she tries to find any motif region in the new sequence using motif libraries such as the PROSITE database, in order to search regions which determine the sequence's function. This search also conducted to predict protein structures (Figure 2.8) (Bork & Koonin, 1996).

Models such as profiles, which can represent categories of sequences accurately, and methods which can learn the parameters of such models speedily are strongly needed and have been actively researched in computational molecular biology over the past several years. When we consider only short sequences as targets, we can say that obtaining profiles and representing motifs lead to a common problem. To solve it, a number of stochastic or statistical models including hidden Markov models in place of available simple motifs, have been proposed to date (Durbin et al., 1998). In this thesis, as well, we define probabilistic networks with finite partitionings to represent motifs (Chapter 4) , and in addition, establish a new learning method for hidden Markov models (Chapter 5).

2.2 Machine Learning

In this section, we first explain the fundamental concepts and terms of the field of machine learning, then describe four types of learning-model classes, each of which is used in a corresponding subsequent chapter, and finally describe three types of algorithms, which are strongly concerned with learning the four model classes (Figure 1.3).

Note that *machine learning* basically refers to all types of automatic knowledge acquisition through the use of computers. Learning from examples, the approach used in this thesis, is merely one way of tackling the basic problems of machine learning, but it is the principal approach used in current research.

2.2.1 Fundamentals of Machine Learning

As briefly mentioned in Chapter 1, the subject of *machine learning* is to provide machines with learning abilities, which has been also one of the major goals of *artificial intelligence* in the field of computer science.

What is Learning?

Since in the early stages of the field of artificial intelligence, the philosophical question 'What is learning ?' has been repeated any number of times and the general answer has been that *learning* is any of various kinds of changes. We introduce some of them here.

Simon (Simon, 1983) characterized the change in detail:

> *Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more effectively the next time.*

Minsky (Minsky, 1986) mentioned that it is merely any useful change:

> *Learning is making useful changes in our mind.*

Michalski (Michalski, 1986) noted that the change has a representation:

> *Learning is constructing or modifying representations of what is being experienced.*

As Michalski defined, we might grasp the meaning of learning as constructing or modifying representations from obtained examples. Here, let us consider a simple example of a learning problem, which classifies given examples into two *classes*, i.e. fruits and vegetables. In this problem, the learner is presented with a finite number of examples, each of which has two *attributes*, the color and weight, and a class. That is, the examples fit the form shown in the following table:

No.	Color	Weight(g)	Class
1	Yellow	20	Fruit
2	Green	28	Vegetable
3	Red	24	Vegetable
4	Orange	17	Fruit
...			

From the examples, the learner may find a rule for classifying the examples into two categories. For instance, when given an example, if its color is yellow or orange and its weight is less than 22 grams, the example is a fruit. The *classification rule* (for short, *rule*) is nothing else but a representation obtained from the examples, as Michalski mentioned, and is a *function* that can be represented as a mapping of a two-dimensional value (the attributes) as input and to a scalar value (the corresponding class of the example) as output. Here, note that the rule is a *deterministic rule* in terms of *prediction*, i.e. that the class of any example is unambiguously determined by the rule.

Further note that in this example, each input has a *label* specifying its class, e.g. a fruit or a vegetable; such learning (with labeled examples) is referred to as *supervised learning.* Conversely, learning with examples which have no labels, is referred to as *unsupervised learning.*

Stochastic Rules

Here, let us consider another example in which we would like to learn a classification rule for predicting whether the weather will be good or not tomorrow based on all the data available until now. In this example, the weather is basically an uncertain event, and it would be impossible to forecast the weather with high accuracy using a deterministic rule. In such situations, it is generally desirable to use rules with assigned probabilities, called *stochastic rules*, to achieve high prediction accuracy. Also, in the cases which can include unexpected errors or noise, learning of a stochastic rule would be preferable (more robust) than learning a deterministic one.

The term *learning* is often replaced by several other terms such as *training*, *estimation* and *fitting.* The term estimation is usually used in the field of *statistical inference.* This term has been used analogously in the machine learning field, because learning stochastic rules from given examples can be regarded as estimating input-output correlations in statistical inference.

Models / Parameters

The term *model* is an extremely elusive term, but generally indicates a form of representation for the rules or functions to be learned from examples. In general, a model includes *parameters* which are to be learned from examples.

We note here that not only the learning of the parameters of a model, but also the selecting a model itself can be an objective of learning. In other words, the *number* of parameters in a model can be learned from examples. This type of learning is referred to as *model estimation* or *model selection*, while learning only the values of the parameters of a model is referred to as *parameter estimation* or *model fitting.*

2.2.2 Model Classes

In this section, we define four classes of models, which are then described in detail in Chapters 3 to 6, in the order presented. They are all stochastic models that learn stochastic rules from given examples. Note that there are numerous other models in use in the machine learning field, including, for example, artificial neural networks.

Stochastic Rule with Finite Partitioning

We define a stochastic rule with finite partitioning which is used to represent a position in α-helix regions in Chapter 3, according to (Mamitsuka & Yamanishi, 1995).

Definition 2.1 (Stochastic Rule with Finite Partitioning) (Mamitsuka & Yamanishi, 1995) Let $\mathcal{X}$ be a countable set, referred to as a *domain.* Let X be a random variable on $\mathcal{X}$. An element $x \in \mathcal{X}$ is referred to as an *instance.* Let $\mathcal{Y}$ be a finite set, referred to as a *label set.* Let Y be a random variable on $\mathcal{Y}$. We call an element $y \in \mathcal{Y}$ a *label.* Let $\{C^{(j)}\}_{1 \le j \le m}$ be a partition of the domain $\mathcal{X}$, called a *finite partitioning* of $\mathcal{X}$, i.e. $\mathcal{X} = \cup_{j=1}^{m} C^{(j)}$ and $C^{(j)} \cap C^{(k)} = \phi$ for $j \neq k$. Each $C^{(j)}$ is called a *cell.* Let $p^{(j)}(y) \in [0,1]$ be a real-valued parameter, where $\sum_{y \in \mathcal{Y}} p^{(j)}(y) = 1$ is assumed for each $j = 1, \cdots, m$. A *stochastic rule with finite partitioning* is a mapping $P : \mathcal{X} \times \mathcal{Y} \to [0,1]$ given as follows:

```
For  $j := 1$ to $m$
  if $x \in C^{(j)}$ and $Y = y$ then
    $P(Y = y \mid X = x) = p^{(j)}(y)$.
```

$\square$

Figure 2.10 shows an example of a stochastic rule with finite partitioning for the two-dimensional domain $\mathcal{X} = \mathcal{X}_a \times \mathcal{X}_b$, where the partition consists of nine cells.

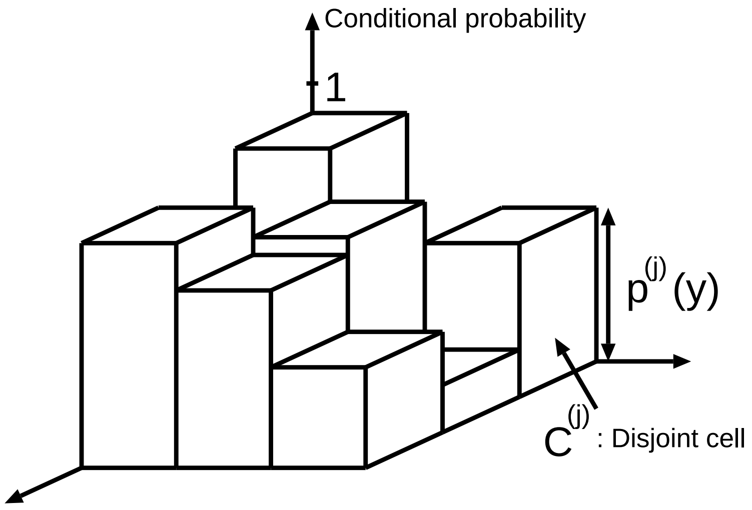


Figure 2.10: A stochastic rule with finite partitioning

In Chapter 3, we define a stochastic rule with finite partitioning for α-helices to determine the conditional probability that given the amino acid at a position, the position is in an α-helix. That is, x is a given amino acid in a test sequence, and for example, $\mathcal{X}_a$ and $\mathcal{X}_b$ are two physico-chemical properties of amino acids, e.g. molecular weight and hydrophobicity. We further established a new algorithm to learn the stochastic rule with finite partitioning, based on the minimum description length (MDL) principle. Note that the MDL learning used in Chapter 3 tries to learn not only the real-valued parameter $p^{(j)}(y)$ but also the number of cells, i.e. m.

Probabilistic Network

Probabilistic networks (or Baysian networks) (Pearl, 1988; Neapolitan, 1989) have been developed in one subfield of artificial intelligence. In Chapter 4, we applied the network to representing inter-residue relations in a motif. We will now define a general probabilistic network.

Let $K = (\mathcal{S}, \mathcal{A})$ be a directed acyclic graph with the node set $\mathcal{S} = \{1, \cdots, n\}$ and the arc set $\mathcal{A}$. If $i \rightarrow j$ is an arc, we say that i is a *predecessor* of j. Let $\pi_i = \{\pi_i^{(1)}, \cdots, \pi_i^{(k_i)}\} \subset \mathcal{S}$ be the set of the predecessors of node i. Note that $i \notin \pi_i$.

A *probabilistic network* consists of a directed acyclic graph $K = (\mathcal{S}, \mathcal{A})$ and n random variables X_i on domains $\mathcal{X}_i$ for $i = 1, \cdots, n$. Let $\mathcal{V} = \{X_1, \cdots, X_n\}$. For a variable X_i, we define the set of *predecessor variables* $\Pi_i = \{X_{\pi_i^{(1)}}, \cdots, X_{\pi_i^{(k_i)}}\} \subset \mathcal{V}$. Let $x_{\pi_i} = (x_{\pi_i^{(1)}}, \cdots, x_{\pi_i^{(k_i)}})$, where $x_{\pi_i^{(j)}}$ is in $\mathcal{X}_{\pi_i^{(j)}}$ for $j = 1, \cdots, k_i$. For a variable X_i and x_{π_i}, the *conditional probability* $p_{z_i|x_{\pi_i}}(i|\pi_i)$ is the probability that X_i is $z_i \in \mathcal{X}_i$ given that $X_{\pi_i^{(j)}}$ is $x_{\pi_i^{(j)}}$ for each $j = 1, \cdots, k_i$. Let x be a sequence $x_1, \cdots, x_n$ of values which are taken on by variables $X_1, \cdots, X_n$, respectively. We denote this sequence by $x_1 \cdots x_n$ for convenience. For x, the *joint probability* $P(x)$ given by K is as follows:

$$P(x) = \prod_{1 \leq i \leq n} p_{x_i|x_{\pi_i}}(i|\pi_i).$$

Figure 2.11 shows a simple example of a probabilistic network consisting of three nodes: 1, 2 and 3. Three random variables X_1, X_2 and X_3 are given to 1, 2 and 3, respectively. The figure shows that the predecessor of node 2 is node 1, and the predecessor of node 3 is also node 1. For a sequence

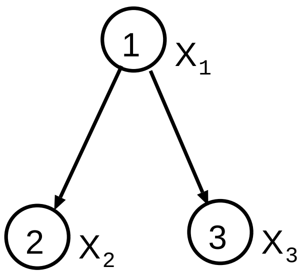


Figure 2.11: A probabilistic network

$x = x_1x_2x_3$, the joint probability for x given by the network in Figure 2.11 is as follows:

$$P(x) = p_{x_1}(1)p_{x_2|x_1}(2|1)p_{x_3|x_1}(3|1).$$

In Chapter 4, we employ a probabilistic network with finite partitionings for representing a short biological sequence, e.g. motif. We will now give the detailed definition of the probabilistic network with finite partitionings.

Definition 2.2 (Probabilistic Network with Finite Partitionings) (Mamitsuka, 1995) A *probabilistic network with finite partitionings* $\mathcal{N}$ consists of the following:
(i) A directed acyclic graph $K = (\mathcal{S}, \mathcal{A})$.
(ii) Random variables X_i on domains $\mathcal{X}_i$ for $i = 1, \cdots, n$.
(iii) A finite partitioning $\{C_i^{(j)}\}_{1 \leq j \leq m_i}$ of $\mathcal{X}_i$ for each $i = 1, \cdots, n$. □

For $x_i \in \mathcal{X}_i$, let $e_i(x_i)$ be the cell number specified by x_i which is given as follows:

```
For  $i := 1$ to $n$
  For  $j := 1$ to $m_i$
    if $x_i \in C_i^{(j)}$ then
      $e_i(x_i) = j$.
```

Let $x = x_1 \cdots x_n$ be an example in the domain $\mathcal{D} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$.
Let $e_{\pi_i}(x_{\pi_i}) = (e_{\pi_i^{(1)}}(x_{\pi_i^{(1)}}), \cdots, e_{\pi_i^{(k_i)}}(x_{\pi_i^{(k_i)}}))$. Let $C_{\pi_i}^{(e_{\pi_i}(x_{\pi_i}))} = (C_{\pi_i^{(1)}}^{(e_{\pi_i^{(1)}}(x_{\pi_i^{(1)}}))}, \cdots, C_{\pi_i^{(k_i)}}^{(e_{\pi_i^{(k_i)}}(x_{\pi_i^{(k_i)}}))})$. For a cell $C_i^{(e_i(x_i))}$ and $C_{\pi_i}^{(e_{\pi_i}(x_{\pi_i}))}$, the *conditional probability* $p_{e_i(x_i)|e_{\pi_i}(x_{\pi_i})}(i|\pi_i)$ is the probability that X_i is in $C_i^{(e_i(x_i))}$ given that $X_{\pi_i^{(j)}}$ is in $C_{\pi_i^{(j)}}^{(e_{\pi_i^{(j)}}(x_{\pi_i^{(j)}}))}$ for each $j = 1, \cdots, k_i$. For x, the *joint probability* $P(x)$ given by $\mathcal{N}$ is as follows:

$$P(x) = \prod_{1 \leq i \leq n} p_{e_i(x_i)|e_{\pi_i}(x_{\pi_i})}(i|\pi_i)$$

Note that the algorithm we will describe in Chapter 4 optimizes the network structure, as well as training the probability parameters. Furthermore, in Chapter 4, a node corresponds to a position in a motif.

Hidden Markov Model (HMM)

We describe a so-called first-order hidden Markov model (hereafter referred to as an HMM) which is typically used for representing biological sequences (Krogh et al., 1994a) and used in Chapter 5.

Definition 2.3 (Hidden Markov Model) A *hidden Markov model* H is a 7-tuple $\langle \Sigma, Q, I, F, E, A, B \rangle$, where:
(i) Σ is a finite alphabet.
(ii) Q is a finite set of *states*.
(iii) $I \subseteq Q$ is a finite set of *initial states*.
(iv) $F \subseteq Q$ is a finite set of *final states*.
(v) $E \subseteq Q \times Q$ is a finite set of *arcs*. An arc $(i, j) \in E$ is also denoted by e_{ij}.
(vi) $A = (a_{ij})_{i,j \in Q}$ is a matrix of *state transition probabilities*, i.e. a_{ij} is the probability that is attached to arc e_{ij} which indicates the transition from state i to j, where $\sum_{j \in Q} a_{ij} = 1$ is assumed for each $i \in Q$.
(vii) $B = (b_j(c))_{j \in Q, c \in \Sigma}$ is a matrix of *symbol output probabilities*, i.e. $b_j(c)$ is the probability that is attached to state j which indicates that state j outputs symbol c, where $\sum_{c \in \Sigma} b_j(c) = 1$ is assumed for each $j \in Q$. □

Possible transitions are made successively from an initial state to a final state, and the relevant transition probability and symbol output probability can be multiplied at each transition to calculate the overall *likelihood* of all the symbols produced in the transition *path* up to that point. When all transitions are finished, the HMM generates a symbol sequence according to the likelihood of each sequence could have been formed. In other words, when the sequence is given, there are one or more transition paths that could have produced it, each of which would generate the sequence with a specific likelihood that the sequence is in the path. We regard the sum of the likelihoods obtained for all such transition paths as the *likelihood that the sequence is generated by the HMM*.

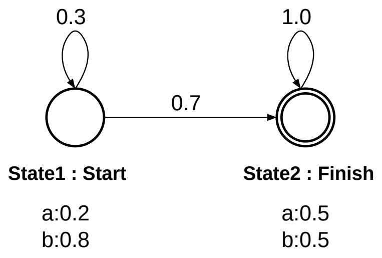


Figure 2.12: A hidden Markov model

Figure 2.12 shows a simple example of an HMM, which contains only two states, i.e. initial state 1 and final state 2, each of which outputs only two symbols, i.e. *a* and *b*. Let us first consider the situation in which the HMM outputs the simple string *aba*. In the HMM, there are only two transition paths, i.e. $1 \rightarrow 1 \rightarrow 2$ and $1 \rightarrow 2 \rightarrow 2$, that could output *aba*. In the first, the *aba* is generated with likelihood $0.0168 (= 0.2 \times 0.3 \times 0.8 \times 0.7 \times 0.5)$. The *aba* is obtained via the second path with likelihood $0.035 (= 0.2 \times 0.7 \times 0.5 \times 1.0 \times 0.5)$. Thus, given the above HMM, the *aba* is generated with the total likelihood $0.0518 (= 0.0168 + 0.035)$.

In general, the likelihood that a sequence is generated by a given HMM is calculated using forward probabilities, defined below.

Definition 2.4 (Forward Probability) (Rabiner, 1989) Let $H = \langle \Sigma, Q, I, F, E, A, B \rangle$ be an HMM. For convenience, let $Q = \{1, \cdots, M\}$ and $\Sigma = \{1, \cdots, L\}$. For a symbol sequence $\sigma = \sigma_1 \cdots \sigma_n$, $1 \leq t \leq n$ and a state j, the *forward probability* $\alpha_\sigma[t, j]$ is the probability that the partial sequence $\sigma_1 \cdots \sigma_t$ is generated, and that the state at time t is j. For $t = 0$ and a state j, the forward probability $\alpha_\sigma[0, j]$ is the probability that no symbol is generated, and that the state at time 0 is j. □

Forward probability $\alpha_\sigma[t, j]$ can be calculated using forward probabilities $\alpha_\sigma[t-1, i]$ $(i = 1, \cdots, M)$, based on dynamic programming approach which is also used in sequence alignment (Algorithm 2.5).

Algorithm 2.5 (Forward)
input: a symbol sequence $\sigma = \sigma_1 \cdots \sigma_n$
output: forward probabilities

```
1.

For $j := 1$ to $M$
  if $j$ is an initial state then
    $\alpha_\sigma[0, j] = 1$,
  else
    $\alpha_\sigma[0, j] = 0$.

2.

For $t := 1$ to $n$
  For $j := 1$ to $M$
    $\alpha_\sigma[t, j] = \sum_{1 \leq i \leq M} a_{ij} b_j(\sigma_t) \alpha_\sigma[t-1, i]$
```

□

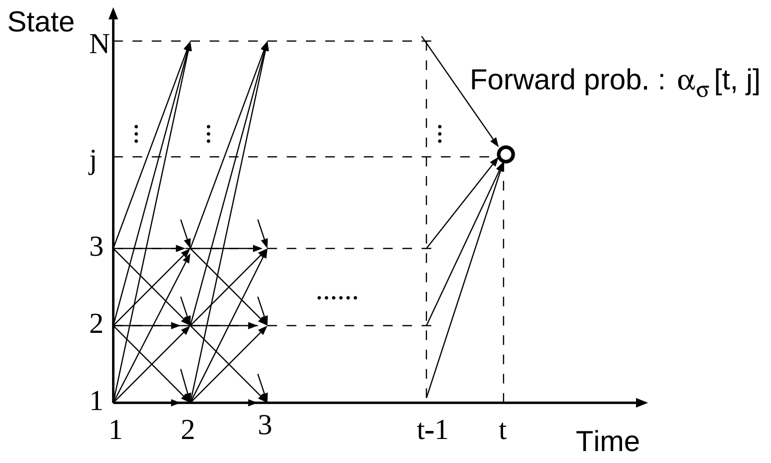


Figure 2.13: Forward probability

This iterative operation leads to a lattice structure as shown in Figure 2.13. In the figure, the recursive computation is repeated for each state and time, to obtain the forward probability $\alpha_t(j)$. From the forward probability $\alpha_\sigma[t, j]$, the likelihood that σ is generated by H, is calculated as $\sum_{1 \leq i \leq M} \alpha_\sigma[n, i]\beta_\sigma[n, i]$, where $\beta_\sigma[n, i] = 1$ if the state i is a final state, and otherwise $\beta_\sigma[n, i] = 0$.

The problem of learning an HMM is to estimate the state transition probabilities a_{ij} ($i = 1, \cdots, M, j = 1, \cdots, M$) and the symbol output probabilities $b_j(c)$ ($j = 1, \cdots, M, c = 1, \cdots, L$) that maximize the likelihood of each training sequence. The most general and conventional algorithm to have been used for the purpose is referred to as the Baum-Welch algorithm, and is a local optimization algorithm which will be described in Section 2.2.3. In contrast with this, we propose a new learning algorithm for HMMs in Chapter 5, which minimizes a kind of error-distance between the real likelihood that a given training sequence would be generated by an HMM and the target likelihood of the sequence. This algorithm allows us to use as training examples not only sequences belonging to the class to be represented by the HMM but also the sequences which do not, e.g. negative examples. In other words, our method can implement supervised learning of HMMs while the Baum-Welch algorithm is based exclusively on unsupervised learning.

When a test sequence whose class is unknown is given, we use the HMM that has been trained by the learning algorithm on a training data set, and parse the input sequence by the trained HMM. In other words, we search the most likely parse of the input string.

Definition 2.6 (Parse by HMM) Let $H = \langle \Sigma, Q, I, F, E, A, B \rangle$ be an HMM. Let σ be a given sequence, $\mathcal{S}$ be a transition path generated by H for σ. Let $P(\sigma, \mathcal{S}|H)$ be the probability that a sequence σ is generated by a path $\mathcal{S}$ in an HMM H. The *most likely parse* is given as follows:

$$arg^1 \ \max_{\mathcal{S}} P(\sigma, \mathcal{S}|H)$$

□

The most likely parse is obtained by replacing '$\sum$' by 'max' in the iterative calculation of the forward probability of Algorithm 2.5, and retaining the most likely sub-parse at any intermediate step.

[1] The $arg \max_x f(x)$ indicates the x which maximizes the f. Similarly, $arg \min_x f(x)$ is the x which minimizes the f.

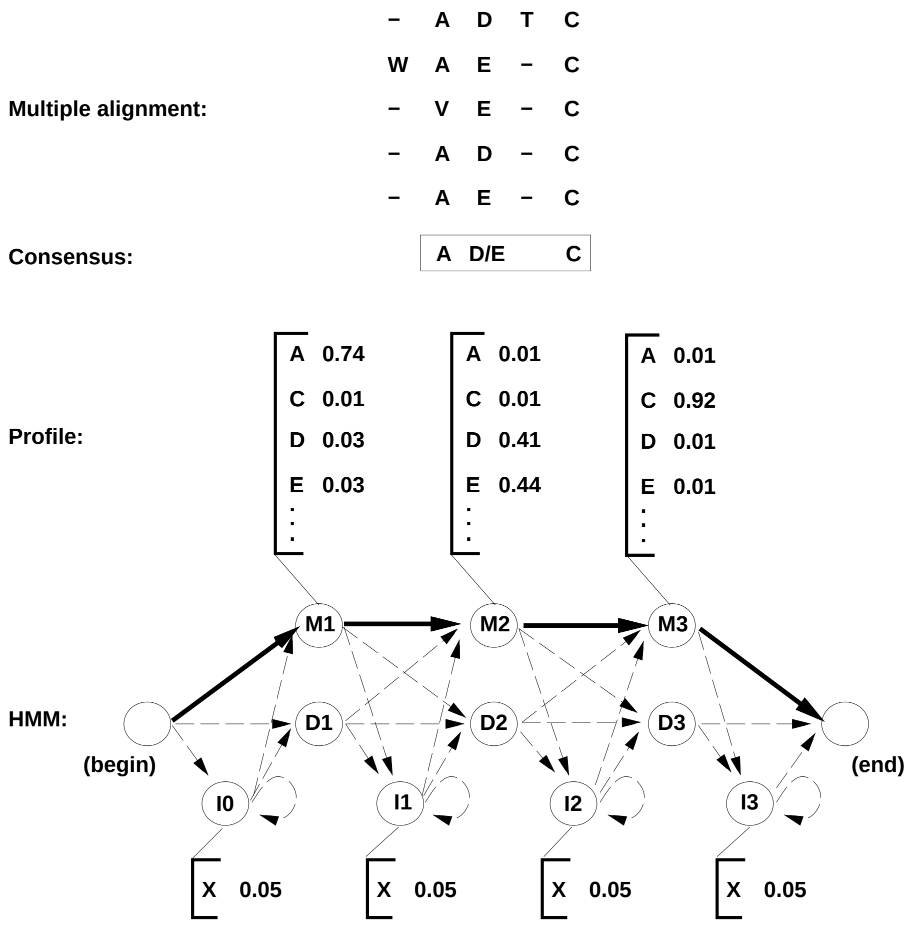


Figure 2.14: A typical hidden Markov model for alignment and profile from (Eddy, Mitchison, & Durbin, 1995)

When we use an HMM to represent an amino acid sequence, we fix the value of M at 20 and N at the approximate length of the sequence. As mentioned earlier, HMMs are used mainly in the problem of aligning multiple sequences and calculating their profile. Figure 2.14 displays a simple example of the HMM used for the problem. Such HMMs are typically referred to as left-to-right models, and have a peculiar structure, in which possible state transitions are limited to flow from states further left to those further right, or to repetition of a given state itself. Furthermore, such HMMs have three types of states: *match*(M), *delete*(D) and *insert*(I). The (M) state outputs one of the 20 amino acids while the delete (D) actually deletes symbols, but also does not output anything, and the insert (I) outputs only a space (gap). In brief, the HMM is designed only for the purpose of aligning multiple sequences.

Figure 2.14 shows that, when five sequences are given as examples to be aligned and the profile of which calculated, the HMM gives symbol output probabilities at M1, M2 and M3, each of which corresponds to A, D/E and C in the consensus sequence, respectively. As in the figure, HMMs have been used both for aligning multiple sequences and calculating their profile at the same time.

Stochastic Ranked Node Rewriting Grammar

A ranked node rewriting grammar (RNRG) (Abe, 1988) is a tree generating system that consists of a single tree structure called the starting tree, and a finite collection of rewriting rules which rewrite a node in a tree with an incomplete tree structure. The node to be rewritten needs to be labeled with a non-terminal symbol, whereas the tree structure can be labeled with both non-terminal and terminal symbols. Here we require that the node being rewritten has the same number of child nodes (called the rank of the node) as the number of empty nodes in the incomplete tree structure. After rewriting, the descendants of the node are attached to these empty nodes in the same order as before rewriting. The tree language generated by an RNRG grammar is the set of all trees whose nodes are labeled solely with terminal symbols that can be generated from the starting tree by a finite number of applications of its rewriting rules. The string language of the grammar is the set of yields from the trees in its tree language, namely the strings that appear on the leaves of the trees, read from left to right.

As a stochastic version of RNRG, we defined a notion of stochastic ranked node rewriting grammar (Abe & Mamitsuka, 1997). Stochastic ranked node rewriting grammars have a richer expressive power than HMMs, since they can deal with long-distance relations contained in a primary sequence, while HMMs represent only relations between adjacent residues (c.f. Figure 2.14). This advantage of tree grammars makes them suitable for representing β-sheet structures retained by hydrogen bonds between amino acids which, while 3-dimensionally adjacent, are mutually distant in the primary sequence. We use ranked node rewriting grammars to represent long-range interactions in a β-sheet and to predict β-sheet structures in a new input sequence. In the natural language context, prediction using grammars corresponds to *parsing* for a new sentence. This use of 'parsing' is synonymous with 'prediction' as used in Chapter 6.

First, we give some preliminary definitions. Let Λ be a set. A directed tree whose nodes are labeled with elements in Λ is called a *tree over* Λ. The *rank of a node* x of a tree over Λ is the number of outgoing edges, denoted by $rank(x)$. A *ranked alphabet* is an alphabet such that each symbol x in the alphabet is assigned a non-negative integer $rank(x)$. A *tree over a ranked alphabet* Λ is a tree α over Λ such that $rank(x) = rank(l(x))$ for each node x, where $l(x)$ denotes the label of x in the tree α. The set of trees over a ranked alphabet Λ is denoted by T_Λ.

We give a detailed definition of an RNRG.

Definition 2.7 (Ranked Node Rewriting Grammar) (Abe & Mamitsuka, 1997) A *ranked node rewriting grammar* G is a 5-tuple $\langle \Sigma_N, \Sigma_T, \sharp, \beta_G, R_G \rangle$, where:
(i) Σ_N is a ranked alphabet called the *non-terminal alphabet* of G.
(ii) Σ_T is a ranked alphabet such that each symbol in Σ_T has rank 0 and is disjoint from Σ_N. The alphabet Σ_T is called the *terminal alphabet* of G.
(iii) $\sharp$ is a distinguished symbol distinct from any member of $\Sigma_N \cup \Sigma_T$, indicating an *empty node*. Let $\Lambda = \Sigma_N \cup \Sigma_T \cup \{\sharp, \lambda\} \cup \{t_i | i = 1, 2, \cdots\}$ be a ranked alphabet, where λ is the empty string, both $\sharp$ and λ have rank 0 and the rank of t_i is i for $i \geq 1$. For convenience, we confuse t_i's and denote them simply by t.
(iv) β_G is a tree in T_Λ such that no node is labeled with $\sharp$. We call β_G the *starting tree* of G.
(v) Let α be a tree in T_Λ. We define the *rank of* α, denoted by $rank(\alpha)$, the number of nodes in α labeled with $\sharp$. A tree α with $rank(\alpha) \geq 1$ is called an *incomplete tree.* A *rewriting rule* of G is a pair $\langle S, \alpha \rangle$ such that S is a non-terminal symbol in Σ_N and α is a tree in T_Λ with $rank(S) = rank(\alpha)$. We write $S \rightarrow \alpha$ for the rewriting rule $\langle S, \alpha \rangle$. Then R_G is a finite set of rewriting rules of G.
We define the *rank of* G by $rank(G) = \max \{rank(S) \mid S \in \Sigma_N\}$. If $k = rank(G)$, then G is called a *rank k grammar.* □

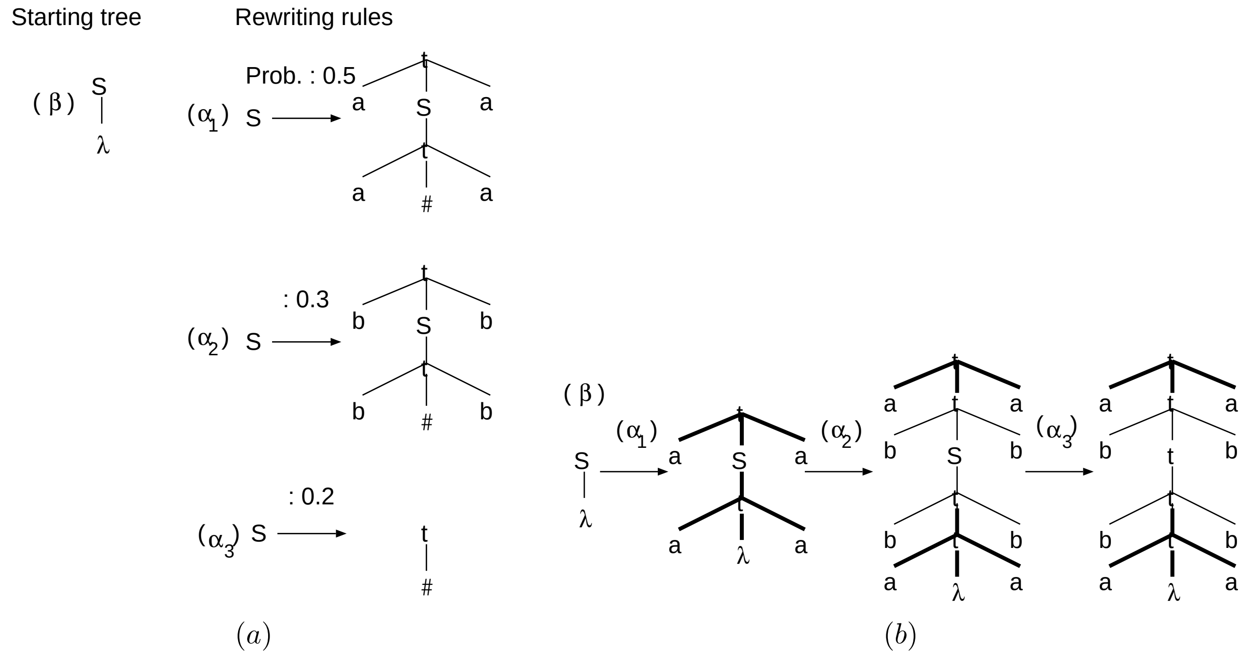


Figure 2.15: (a) A rank 1 grammar G_1 and (b) Derivation of *abbaabba* by G_1

Given an RNRG, a stochastic ranked node rewriting grammar (hereafter referred to as SRNRG) is obtained by assigning to each rewriting rule in the grammar its rule application probability, which is constrained so that for each non-terminal, the sum total of all rule application probabilities for the rewriting rules for that non-terminal equals unity.

Definition 2.8 (Stochastic Ranked Node Rewriting Grammar) (Abe & Mamitsuka, 1997) A *stochastic ranked node rewriting grammar* G is a 6-tuple $\langle \Sigma_N, \Sigma_T, \sharp, \beta_G, R_G, T_G \rangle$, where:
(i) The 5-tuple $\langle \Sigma_N, \Sigma_T, \sharp, \beta_G, R_G \rangle$ is an RNRG.
(ii) For a rewriting rule $r = \langle S_r, \alpha_r \rangle$, $T_G(r)$ is a *rule application probability* of r, where $\sum_{\{r|S_r=S\}} T_G(r) = 1$ is assumed for each $S \in \Sigma_N$. □

We now give an example of an SRNRG. The grammar G_1, shown in Figure 2.15(a) is a rank 1 grammar and defines a probability distribution over the language $L_1 = \{ww^Rww^R|w \in \{a,b\}\}$, where S is a non-terminal symbol, a, b are terminal symbols and ww^R is the notation for a word (i.e. symbol-string) followed by its reverse. The process of derivation by an SRNRG is illustrated in Figure 2.15(b), which exhibits the derivation of the string *abbaabba* by G_1, generated with likelihood $0.5 \times 0.3 \times 0.2 = 0.03$. Note that the language L_1 captures the type of dependency existing in the amino acid sequence pattern of 'anti-parallel' four-strand β-sheets, that is, *..ab..ba..ab..ba..*, where the use of an identical letter indicates that these positions face each other in a β-sheet structure, and are believed to be correlated.

In this thesis, we propose the 'linear' subclass of SRNRGs, called the linear SRNRG, that is, the subclass of SRNRG for which each rewriting rule contains at most one node labeled with a non-terminal symbol of rank greater than 0. This constraint significantly simplifies the prediction and learning algorithms for SRNRGs. The example shown in Figure 2.15 corresponds to this linear subclass. In the linear SRNRGs, we further assume that each leaf represents a distribution referred to as 'symbol generation probabilities' or 'amino acid generation probabilities' over the set of 20 letters corresponding to the 20 amino acids. Thus, using two types of probabilities, i.e. rule application probabilities and symbol generation probabilities, we can calculate the likelihood of a sequence given an RNRG by multiplying them, just we did as the likelihood of a sequence given an HMM.

2.2.3 Learning Algorithms

We review three types of learning algorithm: maximum likelihood (ML) estimation, minimum description length (MDL) learning, and the Baum-Welch algorithm for HMMs. The maximum likelihood is the most basic approach for estimating model parameters, and we should mention it first since it is related to the minimum description length learning, which is derived from information theory and is applied to a stochastic rule with finite partitioning and a probabilistic network with finite partitionings in Chapters 3 and 4, respectively. The Baum-Welch learning algorithm, which is a conventional algorithm for HMMs and an extended version of which is established for learning SRNRGs in Chapter 6, is based on the maximum likelihood criterion. Furthermore, in Chapter 5, the Baum-Welch algorithm is compared with our new supervised learning algorithm as one of conventional algorithms for HMMs. We note that the maximum likelihood approach and the Baum-Welch algorithm based on it are used only for parameter estimation in a given model, while minimum description length learning is used in determination of the model itself.

Maximum Likelihood (ML) Estimation

Maximum likelihood (hereafter referred to as ML) is the simplest and most fundamental learning criterion. In it, broadly speaking, the parameters of a model are calculated so that the likelihood of the target data given the model is maximized.

Definition 2.9 (Maximum Likelihood Estimator)
Let $\mathcal{X}$ be a probability space with an unknown probability distribution P^* over $\mathcal{X}$. Let $x^n = x_1 \cdots x_n$ be a sequence of n elements generated according to P^*. Let $P(\theta)$ be a probability distribution over $\mathcal{X}$ with a real-valued parameter vector θ. Let Θ be a set of all real-valued parameter vectors of $P(\theta)$. Let $P(x^n : \theta)$ be the probability that x^n is generated given $\theta \in \Theta$. The *maximum likelihood estimator* $\hat{\theta}$ for P^* is given as follows:

$$\hat{\theta} = arg \max_{\theta \in \Theta} P(x^n : \theta). \tag{2.1}$$

□

Here, let us consider a simple example.

Example 2.10 (Histogram Density) (Hall & Hannan, 1988) Let $\mathcal{X}$ be $[0,1]$, and $x \in \mathcal{X}$ is called an instance. Let $\{C_j\}_{1 \leq j \leq m+1}$ be a partition of $\mathcal{X}$, and each C_j is called a *cell*. Let $\theta_j \in [0,1] (j = 1, \cdots, m+1)$ be a real-valued parameter, where $\sum_{1 \leq j \leq m+1} \theta_j = 1$ is assumed. A *histogram density* is a mapping of $P : \mathcal{X} \to [0, m+1]$ given by the following form:

```
For  $j := 1$ to $m+1$
  if $x \in C_j$ then
    $P(x) = (m+1)\theta_j$,
```

Let n_i be the number of examples in the i-th cell C_i. The likelihood of the given examples being produced by the parameter values θ_i $(i = 1, \cdots, m+1)$ is:

$$\prod_{1 \le i \le m+1} ((m+1)\theta_i)^{n_i} = (m+1)^n \prod_{1 \le i \le m+1} \theta_i^{n_i}, \tag{2.2}$$

where $n = \sum_{1 \le i \le m+1} n_i$. From Eq. (2.2), the maximum likelihood estimator of θ_i should be set as follows:

$$\begin{aligned}\hat{\theta}_i &= \frac{n_i}{\sum_{1 \le i \le m+1} n_i} \\ &= \frac{n_i}{n} \ (i = 1, \cdots, m+1).\end{aligned}$$

□

Minimum Description Length (MDL) Learning

The minimum description length principle (hereafter referred to as MDL) is derived from the literature of information theory. Here, first we briefly review the idea behind the MDL principle.

Let us consider a problem in which an encoder A has to communicate with a decorder B. Here suppose that $\mathcal{X} = \{0, 1\}$, and $x^n = x_1 \cdots x_n$, a given set of data, is a binary sequence, where $x_i \in \mathcal{X}$. Here A would like to encode the sequence x^n and send it to B assuming no error or noise. Of course, the sequence can be sent with n bits, but A would like to send it with fewer. Under the circumstances, A first encodes the sequence with a probability distribution P for $x \in \mathcal{X}$, i.e. $P(x^n)$ and sends it to B. Next, A encodes P itself, and then sends it to B. If B receives both messages, B first decodes P and then obtains x^n using it. Therefore, A and B require two processes each for encoding and decoding, respectively, and the best method, which can communicate with the minimum number of bits from A to B, must minimize the total number of bits that must be processed by the two steps. If encoding P requires $C_n(P)$ bits, this requirement is expressed as follows:

$$\min_P \{-\log P(x^n) + C_n(P)\}.$$

This rough idea can be expanded into a principle for estimating an optimal probability distribution from n given instances (Rissanen, 1978; Barron & Cover, 1991; Yamanishi, 1992).

We here give some basics of information theory. Let $\mathcal{X}$ be a countable set. Let E be a function $E : \mathcal{X} \to \{0, 1\}^*$. We call E an *encoding* function and $D = E^{-1}$ a *decoding* function. If we encode a sequence $x^n = x_1 \cdots x_n$ with $x_i \in \mathcal{X}$ $(i = 1, \cdots, n)$ into $y^n = y_1 \cdots y_n$ with $y_i = E(x_i)$ $(i = 1, \cdots, n)$, then we call x the *source sequence*, y the *code sequence* and each y_i the *codeword*. Let $l(x)$ be the length of $E(x)$, i.e. the number of bits for encoding x by a function E, called a *code-length*. Let $v, w, z \in \{0, 1\}^*$. We call v a *prefix* of w if there is z such that $w = vz$. A set $A \subseteq \{0, 1\}^*$ is *prefix-free*, if no element in A is the prefix of another element in A. An encoding function E defines a *prefix-code*, if the set of codewords generated by an encoding function E is prefix-free. Note that when we encode $x \in \mathcal{X}$ with a prefix-code, the code-length $l(x)$ satisfies the following Kraft inequality (Kraft, 1949):

$$\sum_{x \in \mathcal{X}} 2^{-l(x)} \le 1.$$

We define the *description length* for $x \in \mathcal{X}$ as the number of bits for encoding x with a prefix-code. Let P be a probability distribution for $x \in \mathcal{X}$, where $\sum_{x \in \mathcal{X}} P(x) = 1$ is assumed. Let L be the *average codeword length* given as $L = \sum_{x \in \mathcal{X}} P(x) l(x)$. We write L_{min} for the minimum of L.

Theorem 2.11

$$L_{min} \geq H(P) = \sum_{x \in \mathcal{X}} P(x) \log P(x), \tag{2.3}$$

where L_{min} is obtained when $l(x) = -\log P(x)$.
(Proof)
From the following Lagrangian function

$$\mathcal{L}(l(x), P(x)) \quad = \quad \sum_{x \in \mathcal{X}} P(x) l(x) + \lambda(\sum_{x \in \mathcal{X}} 2^{-l(x)} - 1),$$

$\sum_{x \in \mathcal{X}} P(x) l(x)$ is minimized when $\lambda = \frac{1}{\ln 2}$ and $l(x) = -\log P(x)$. □

The theorem indicates that when a probability distribution P for $x \in \mathcal{X}$ is given, the number of bits required for encoding x with a prefix-code is given as $-\log P(x)$.

Definition 2.12 (Minimum Description Length Estimator) (Rissanen, 1978, 1989)
Let $\mathcal{X}$ be a probability space with an unknown probability distribution P^* over $\mathcal{X}$. For simplicity, $\mathcal{X}$ is countable. Let $x^n = x_1 \cdots x_n$ be a sequence of n elements generated according to P^*. Let P be a probability distribution over $\mathcal{X}$. Let $\mathcal{H}$ be a set of all probability distributions over $\mathcal{X}$. For a sequence x^n, the number of bits for encoding x^n with a prefix-code is given as $-\log P(x^n)$. Let C_f be an encoding function $C_f : \mathcal{H} \rightarrow \{0,1\}^*$, which defines a prefix-code. Let $C_n(P)$ be the number of bits for encoding P with a prefix-code defined by C_f, given x^n. The *minimum description length estimator* $\hat{P}$ for P^* is given as follows:

$$\hat{P} = arg \min_{P \in \mathcal{H}} \{-\log P(x^n) + C_n(P)\}. \tag{2.4}$$

□

Note that the description length can be divided into two parts in all cases. The MDL principle given by Eq. (2.4) can be applied directly to any of the probabilistic models used in various types of learning problems. When learning using the equation, the first term of the description length indicates how the model specified by P fits the given examples, and the second term can be regarded as a kind of penalty which depends on the complexity of the model. Therefore, broadly speaking, this principle can solve the major trade-off in probabilistic models, which arises because simple rules cannot express given training examples exactly and will not achieve good results on test data, while complicated rules are often weak in accounting for new data though they exactly fit to the training data given. Under this trade-off, the MDL principle gives us a moderately complex rule with assured optimality, as has been described in the information theory literature. Thus, we can use the MDL principle to optimize the probabilistic model (i.e. number of parameters) itself, rather than merely to estimate the values of probability parameters in the model.

We give an example of calculating the description length of the models to which the MDL principle can be applied. In this example, we determine the number of cells for the histogram density which we described in Definition 2.10, and we here use the notation defined there.

Example 2.13 (Description Length of Histogram Density) (Rissanen, 1989) The first term of Eq. (2.4), which by itself corresponds to the maximum likelihood estimation, is easily derived from Eq. (2.2) as follows :

$$-\log P(x^n) = -\sum_{1 \leq i \leq m+1} n_i \log \frac{n_i}{n} - n \log(m+1)$$

The second term of Eq. (2.4) is given as follows:

$$\frac{m \log n}{2},$$

since each parameter θ_i in Eq. (2.2) can be estimated with accuracy of $O(\frac{1}{\sqrt{n}})$.

Thus, the MDL principle tells us that the m which minimizes the following formula is the optimal number of cells for the histogram densities.

$$- \sum_{1 \leq i \leq m+1} n_i \log \frac{n_i}{n} - n \log(m+1) + \frac{m \log n}{2} \tag{2.5}$$

□

Baum-Welch algorithm for HMMs

The Baum-Welch algorithm is a conventional learning algorithm for probability parameters of HMMs, i.e. state transition probabilities and symbol output probabilities, and it will serve as the basis for our learning algorithm for SRNRGs. This algorithm practically implements *maximum likelihood* estimation on HMMs (Rabiner, 1989). Note that only the parameters of an HMM are trained by this algorithm: the HMM structure itself is initially given. This is in contrast with MDL learning which optimizes the structure itself.

In addition to the forward probability $\alpha_\sigma[t, j]$ defined in Definition 2.4, we define here the backward probability.

Definition 2.14 (Backward Probability) (Rabiner, 1989) Let $H = \langle \Sigma, Q, I, F, E, A, B \rangle$ be an HMM. For convenience, let $Q = \{1, \cdots, M\}$ and $\Sigma = \{1, \cdots, L\}$. For a symbol sequence $\sigma = \sigma_1 \cdots \sigma_n$, $0 \leq t \leq n-1$ and a state i, the *backward probability* $\beta_\sigma[t, i]$ is the probability that the partial sequence $\sigma_{t+1} \cdots \sigma_n$ is generated, and that the state at time t is i. For $t = n$ and a state i, the backward probability $\beta_\sigma[n, i]$ is the probability that no symbol is generated, and that the state at time n is i. □

The backward probability can be also calculated iteratively, based on the dynamic programming approach, as follows:

Algorithm 2.15 (Backward)
input: a symbol sequence $\sigma = \sigma_1 \cdots \sigma_n$
output: backward probabilities

```
1.

For $i := 1$ to $M$
  if $i$ is an final state then
    $\beta_\sigma[n, i] = 1$
  else
    $\beta_\sigma[n, i] = 0$

2.

For $t := n-1$ to $0$
  For $i := 1$ to $M$
    $\beta_\sigma[t, i] = \sum_{1 \leq j \leq M} a_{ij} b_j(\sigma_{t+1}) \beta_\sigma[t+1, j]$
```

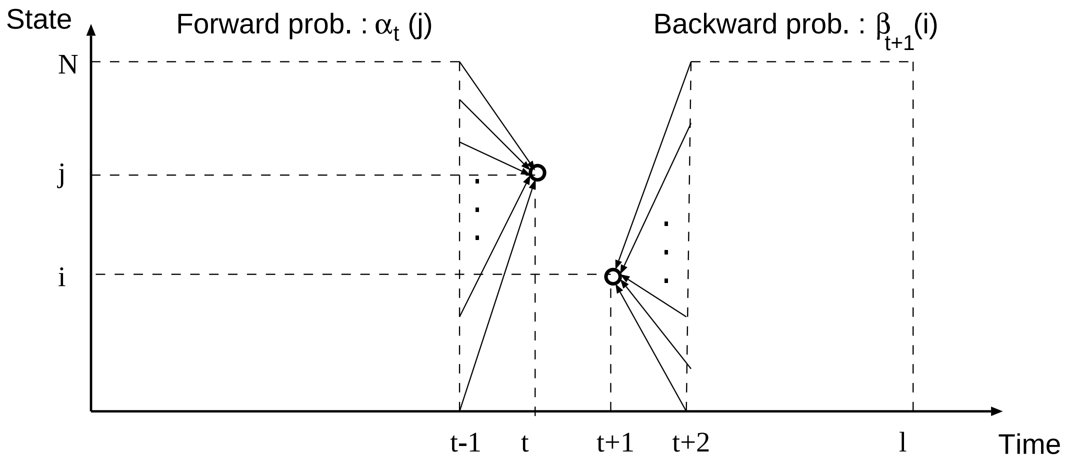

State
Forward prob. : α_t (j)
Backward prob. : β_{t+1}(i)
N
j
i
t-1
t
t+1
t+2
l
Time

Figure 2.16: Forward and backward probabilities

□

The backward as well as the forward probability can be illustrated in Figure 2.16. From these forward and backward probabilities, the likelihood of a training sequence σ given a model H, i.e. $P(\sigma|H)$, is given as $\sum_{1\le i\le M}\alpha_\sigma[n,i]\beta_\sigma[n,i]$ (and $\sum_{1\le i\le M}\alpha_\sigma[0,i]\beta_\sigma[0,i]$).

We define the following two probabilities ξ and γ.

Definition 2.16 (ξ and γ) (Rabiner, 1989) Let $H=\langle\Sigma,Q,I,F,E,A,B\rangle$ be an HMM. For a symbol sequence $\sigma=\sigma_1\cdots\sigma_n$, $0\le t\le n-1$ and two states i and j, $\xi_\sigma[t,i,j]$ is the probability that σ is generated by H, and that the two states at times t and $t+1$ are i and j, respectively. Similarly, for a symbol sequence $\sigma=\sigma_1\cdots\sigma_n$, $0\le t\le n$ and a state i, $\gamma_\sigma[t,i]$ is the probability that σ is generated by H, and that the state at time t is i. □

The probabilities ξ and γ are calculated using the forward and backward probabilities as follows:

$$\begin{aligned}\xi_\sigma[t,i,j] &= \frac{\alpha_\sigma[t,i]a_{ij}b_j(\sigma_{t+1})\beta_\sigma[t+1,j]}{P(\sigma|H)} \quad (0\le t\le n-1),\\ \gamma_\sigma[t,i] &= \frac{\alpha_\sigma[t,i]\beta_\sigma[t,i]}{P(\sigma|H)} \quad (0\le t\le n).\end{aligned}$$

Using ξ and γ, we describe the procedures, known as the *Baum-Welch* (or the *Forward and Backward*) algorithm, used to update a_{ij} and $b_i(c)$. The goal of the algorithm is to maximize the likelihood of the observed symbol sequence σ when given the model H, that is, $P(\sigma|H)$. The algorithm is guaranteed to find a local optimal for the maximum likelihood settings of the probability parameters of an HMM. The probability $\hat{a}_{ij}$, below, is re-estimated using γ and ξ, as the expected proportion of the transitions made from state i that have state j as their destination among all the transitions made from state i. Similarly, $\hat{b}_i(c)$ is re-estimated using γ, as the expected proportion of the symbols generated at state i that are c as opposed to any other symbols (Algorithm 2.17).

Algorithm 2.17 (Baum-Welch) (Rabiner, 1989)
input: a symbol sequence σ and initial a_{ij} and $b_j(c)$
output: trained a_{ij} and $b_j(c)$

Repeat the following step until a stopping condition is satisfied, usually until the changes in the probabilities become smaller than a certain preset amount.
1: Re-estimate the a_{ij} and $b_j(c)$ using training sequences, Eqs. (2.6) and (2.7).

$$\hat{a}_{ij} = \frac{\sum_{0\leq t\leq n-1} \xi_\sigma[t,i,j]}{\sum_{0\leq t\leq n-1} \gamma_\sigma[t,i]}, \tag{2.6}$$

$$\hat{b}_i(c) = \frac{\sum_{1\leq t\leq n|\sigma_t=c} \gamma_\sigma[t,i]}{\sum_{1\leq t\leq n} \gamma_\sigma[t,i]}. \tag{2.7}$$

□

When more than one symbol sequences are given to be trained, each summation of Eqs. (2.6) and (2.7) is summed over the sequences.

When we apply this algorithm to SRNRGs, the state and transition probability of the HMMs are replaced with the rewriting rule and rule application probability in the tree grammer, respectively. Furthermore, instead of the symbol output probability in HMMs, the symbol generation probability distribution of each leaf node is used in the algorithm as applied to SRNRGs.

Derivation of the Baum-Welch Re-estimation Algorithm

1. EM Algorithm and Q Function

Let $\mathcal{X}$ be a countable set, and let X be a random variable on $\mathcal{X}$. Let $\mathcal{Y}$ be a countable set, and let Y be a random variable on $\mathcal{Y}$. Assume that there exists a mapping $y \in \mathcal{Y} \rightarrow x \in \mathcal{X}$, and x is observable, but y is not. The x and y are referred to as *incomplete data* and *missing data*, respectively, and a pair (x, y) is referred to as *complete data*.

Let $P(\theta)$ be a probability distribution with a real-valued parameter vector θ over a probability space. Let Θ be a set of all real-valued parameter vectors. Let $f(x|\theta)$ be the likelihood that x is generated over $\theta \in \Theta$. Similarly, let $f(x,y|\theta)$ be the likelihood that both x and y are generated over $\theta \in \Theta$ and let $f(y|x,\theta)$ be the likelihood that y is generated over $\theta \in \Theta$ given that x is generated. Note the following relation:

$$f(x|\theta) = \frac{f(x,y|\theta)}{f(y|x,\theta)}$$

The purpose of the EM (Expectation Maximization) algorithm is to maximize the logarithmic likelihood $\log f(x|\theta)$ over $\theta \in \Theta$.

Taking the conditional expectation over $y \in \mathcal{Y}$ for the logarithmic likelihood $\log f(x|\Theta)$, we obtain

$$\log f(x|\theta) = \sum_y f(y|x,\theta) \log f(x,y|\theta) - \sum_y f(y|x,\theta) \log f(y|x,\theta). \tag{2.8}$$

Here, we define the function Q and the function H for parameters θ and $\bar{\theta}$, as follows:

Definition 2.18 (Q Function and H Function)

$$Q(\theta|\bar{\theta}) = \sum_y f(y|x,\theta) \log f(x,y|\bar{\theta}) \tag{2.9}$$

$$H(\theta|\bar{\theta}) = \sum_y f(y|x,\theta) \log f(y|x,\bar{\theta})$$

□

Using the definitions, we can rewrite Eq.(2.8) as follows:

$$\log f(x|\theta) = Q(\theta|\theta) - H(\theta|\theta)$$

For any two parameters θ and $\bar{\theta}$, we consider the difference of the logarithmic likelihood,

$$\log f(x|\theta) - \log f(x|\bar{\theta}) = \{Q(\theta|\theta) - Q(\theta|\bar{\theta})\} - \{H(\theta|\theta) - H(\theta|\bar{\theta})\}. \tag{2.10}$$

Here, note that $-\{H(\theta|\theta) - H(\theta|\bar{\theta})\} \leq 0$ with equality if and only if $\theta = \bar{\theta}$.

Lemma 2.19

$$-\{H(\theta|\theta) - H(\theta|\bar{\theta})\} \leq 0$$

(Proof)

$$\begin{aligned} H(\theta|\bar{\theta}) - H(\theta|\theta) &= \sum_y f(y|x,\theta) \log \frac{f(y|x,\bar{\theta})}{f(y|x,\theta)} \\ &\leq \sum_y f(y|x,\theta)\{\frac{f(y|x,\bar{\theta})}{f(y|x,\theta)} - 1\} \quad \text{since} \quad \log z \leq z - 1 \ (0 \leq z \leq 1) \\ &= \sum_y f(y|x,\bar{\theta}) - \sum_y f(y|y,\theta) \\ &= 0 \end{aligned}$$

□

Theorem 2.20 (EM)

$$Q(\theta|\bar{\theta}) \geq Q(\theta|\theta) \quad \Rightarrow \quad f(x|\bar{\theta}) \geq f(x|\theta)$$

(Proof)
Using Eq. (2.10) and Lemma 2.19, this theorem holds. □

This result leads to the following EM algorithm to maximize $f(x|\theta)$.

Algorithm 2.21 (EM)
input: training data
output: trained parameters
1. Choose an initial parameter θ and repeat the following steps 2 and 3.
2. **E-step:** Given θ, compute $Q(\theta|\bar{\theta})$.
3. **M-step:** Choose $\hat{\theta} = arg\max_{\bar{\theta}} Q(\theta|\bar{\theta})$, and set $\theta = \hat{\theta}$. □

2. Hidden Markov Model and Baum-Welch Re-estimation

In hidden Markov models, the x and y correspond to a given symbol sequence and a transition path of states (i.e. a state sequence), respectively. Let $\sigma = \sigma_1 \cdots \sigma_n$ be a given symbol sequence and $S = S_1 \cdots S_n$ be a state sequence.

Definition 2.22 (R Function)

$$R(\theta|\bar{\theta}) = \sum_S f(\sigma, S|\theta) \log f(\sigma, S|\bar{\theta}) \tag{2.11}$$

□

Using the definition, we can write Eq.(2.10) as follows:

$$Q(\theta|\bar{\theta}) = \frac{1}{f(\sigma|\theta)} R(\theta|\bar{\theta}) \tag{2.12}$$

Here, let $H = \langle \Sigma, Q, I, F, E, A, B \rangle$ be an HMM. For convenience, let $Q = \{1, \cdots, M\}$ and $\Sigma = \{1, \cdots, L\}$. Then, using the state transition probabilities a_{ij} $(i = 1, \cdots, M, j = 1, \cdots, M)$ and symbol output probabilities $b_j(c)$ $(j = 1, \cdots, M, c = 1, \cdots, L)$, we can write $\log f(\sigma, S|\bar{\theta})$ as follows:

$$\log f(\sigma, S|\bar{\theta}) = \sum_{1 \le t \le n-1} \log \bar{a}_{S_t S_{t+1}} + \sum_{1 \le t \le n} \log \bar{b}_{S_t}(\sigma_t) \tag{2.13}$$

Substituting Eq. (2.13) into Eq. (2.11), the function R can be seen as follows:

$$\begin{aligned}
R(\theta|\bar{\theta}) &= \sum_S f(\sigma, S|\theta) \sum_t \log \bar{a}_{S_t S_{t+1}} + \sum_S f(\sigma, S|\theta) \sum_t \log \bar{b}_{S_t}(\sigma_t) \\
&= \sum_{S_1} \cdots \sum_{S_n} f(\sigma, S_1, \cdots, S_n|\theta) \sum_t \log \bar{a}_{S_t S_{t+1}} + \sum_{S_1} \cdots \sum_{S_n} f(\sigma, S_1, \cdots, S_n|\theta) \sum_t \log \bar{b}_{S_t}(\sigma_t) \\
&= \sum_t \sum_{S_t} \sum_{S_{t+1}} \log \bar{a}_{S_t S_{t+1}} \sum_{S_1} \cdots \sum_{S_{t-1}} \sum_{S_{t+2}} \cdots \sum_{S_n} f(\sigma, S_1, \cdots, S_n|\theta) \\
&+ \sum_t \sum_{S_t} \log \bar{b}_{S_t}(\sigma_t) \sum_{S_1} \cdots \sum_{S_{t-1}} \sum_{S_{t+1}} \cdots \sum_{S_n} f(\sigma, S_1, \cdots, S_n|\theta) \\
&= \sum_t \sum_i \sum_j \log \bar{a}_{ij} \sum_{S_1} \cdots \sum_{S_{t-1}} \sum_{S_{t+2}} \cdots \sum_{S_n} f(\sigma, S_1, \cdots, S_{t-1}, S_t = i, S_{t+1} = j, S_{t+1}, \cdots, S_n|\theta) \\
&+ \sum_t \sum_j \sum_c \log \bar{b}_j(c) \sum_{S_1} \cdots \sum_{S_{t-1}} \sum_{S_{t+1}} \cdots \sum_{S_n} f(\sigma, S_1, \cdots, S_{t-1}, S_t = j, S_{t+1}, \cdots, S_n|\theta) \\
&= \sum_i \sum_j \log \bar{a}_{ij} \sum_t \xi_\sigma[t, i, j] + \sum_j \sum_c \log \bar{b}_j(c) \sum_{\{t|\sigma_t = c\}} \gamma_\sigma[t, i]
\end{aligned} \tag{2.14}$$

Substituting Eq. (2.14) into Eq. (2.12), the function Q can be seen as follows:

$$Q(\theta|\bar{\theta}) = \sum_i \sum_j \log \bar{a}_{ij} \cdot \frac{\sum_t \xi_\sigma[t, i, j]}{f(\sigma|\theta)} + \sum_j \sum_c \log \bar{b}_j(c) \cdot \frac{\sum_{\{t|\sigma_t = c\}} \gamma_\sigma[t, i]}{f(\sigma|\theta)} \tag{2.15}$$

From Eq. (2.15), we obtain Eqs. (2.6) and (2.7) in Algorithm 2.17 by choosing $\hat{\theta} = arg \max_{\bar{\theta}} Q(\theta|\bar{\theta})$ in Eq. (2.15).

Chapter 3

Predicting α-helices Based on Stochastic Rule Learning

3.1 Introduction

Predicting the protein 3-dimensional structure of a given amino acid sequence is the most crucial problem in the field of computational molecular biology. As such, it has been extensively studied. In particular, a number of approaches have been proposed since the 1970s for the problem of predicting protein secondary structures, which problem is generally placed as the first step to understand the overall 3-dimensional structure of a given target sequence. These approaches have been applied to the *three-state prediction problem*, which is defined as assigning one of the three labels, i.e. α-helix, β-sheet and others, of secondary structures to each of the amino acids in the input sequence. However, the accuracy in predicting such a problem was at most 70% even in the midst of 1990s, an unsatisfactory level for the future purpose of predicting the 3-dimensional structure of a given protein.

These approaches have other disadvantages in addition to the limits of their prediction ability. In the feed-forward type neural network frequently used by the methods proposed in the late 1980s or early 1990s (Qian & Sejnowski, 1988; Kneller et al., 1990; Zhang et al., 1992; Stolorz et al., 1992; Hayward & Collins, 1992), for example, it is difficult for us to find any comprehensive relation between a *region*, i.e. a subsequence of amino acids, predicted by the network and the region used for training the network, since such relations are simply represented by numerical weights in the trained network.

On the other hand, Muggleton et al. provided a method for constructing a rule through which we can observe relations between a region predicted by a rule and the region used for training the rule (Muggleton, King, & Sternberg, 1992). Their rule takes a rough deterministic form, as, for instance, "in an α-helix, the amino acids at the second position must be large ones, and positive amino acids must appear at the next position," etc. This kind of deterministic rule, however, does not always reflect the actual amino acid distribution at a given position of a secondary structure because the categories used for specifying amino acids (e.g. "large", "positive", etc.) are not accurate enough and more exact specification, such as a probability distribution over amino acids, is required.

It is in response to these various drawbacks that we proposed an original method (hereafter referred to simply as the SR method, for stochastic rules) for predicting secondary structures (Mamitsuka & Yamanishi, 1992, 1993, 1995). In this method, we define, as a representation of a probability distribution of amino acids, a stochastic rule for each residue position. In order to learn a stochastic rule, we establish a novel algorithm to optimally categorize twenty amino acids based on the minimum description length principle derived from the information theory field, using the physico-chemical properties of amino acids themselves, into roughly ten to fifteen groups. We focus on the problem of predicting whether any given region in a sequence is α-helix or not. We call this problem the *two-state prediction problem.*

We addressed this particular issue from the following two reasons:

1. *Crucial secondary structure.*
   α-helix is the local structure that has been most extensively investigated in biochemistry and has turned out to be the most crucial among the three states. Thus, predicting α-helices in a given protein sequence is extremely important in understanding protein structures.

2. *Prediction from local properties.*
   While the α-helix is determined to a high degree by local properties alone, such as the short-range interactions of residues, the β-sheet structure, which is another secondary structure to be predicted in the three-state prediction problem, is significantly affected, as well, by long-range interactions, which extend far beyond the range of local properties. Despite this fact, the secondary structure prediction methods which had been proposed in the same period of the SR method, tried to predict the secondary structures by using the local properties of a given sequence only, and they achieved at most 70% average three-state prediction accuracy. We want to find out how accurately we could predict secondary structures from the local properties alone if we limited our prediction to only the α-helix.

We now describe the work of several groups which studied the two-state prediction problem before and when we proposed the SR method. Bohr et al. (1988) and Petersen et al. (1990) applied the same neural network learning method as the Qian and Sejnowski's method (hereafter referred to as QS) to this problem and achieved 72% prediction accuracy for rhodopsin only. Kneller et al. (1990) studied the problem with a neural network based approach. They achieved 79% average prediction accuracy for 22 α-domain type proteins, but their test was based on the cross-validation method using these proteins, some of which have more than 40% pairwise sequence similarity. In such a dataset in which some two proteins have more than 25% pairwise sequence similarity, secondary structure prediction is required no longer, since existing methods for predicting 3-dimensional structures such as homology modeling can be practically applied to such a dataset to predict the exact protein 3-dimensional structure of a test sequence. Muggleton et al. (1992) predicted α-helices of α-domain type proteins using inductive logic programming with 80% average prediction accuracy, but they used only four proteins as test and the sequence similarities among training and test data were not described. Hayward and Collins (1992) addressed the problem of α-helix prediction with the same data as those used by Qian and Sejnowski and achieved 78% average accuracy, but the sequence similarities among the data were not clearly stated and the α-helix content in the test sequences was only 28%. Rost and Sander (1993b) (hereafter referred to as RS) also addressed the two-state prediction problem by neural networks using multiple sequence alignment to enhance the training data. They achieved 80% prediction accuracy for a dataset in which any test sequence has less than 25% pairwise sequence similarity to any training sequence, but they used only α-domain type proteins for test sequences.

In our research, we attempted to predict α-helices in a number of proteins belonging to a wide variety of types, each of which possesses less than 25% pairwise sequence homology to any protein used for training. We emphasize here that the proteins used by the SR method for prediction were not limited to α-domain type proteins, while Kneller et al.'s, Muggleton et al.'s and Rost and Sander's methods were all applied to α-domain type proteins only.

We summarize a number of characteristics of the SR method and our experiments.

1) *Probabilistic region prediction.*
   The SR method predicts the probability that a given test region is an α-helix while most conventional methods try to predict that a given residue position is in one of three secondary structures,

i.e. α-helix, β-sheet and others. We call this probability a likelihood. The idea of region prediction was also reported by Presnell et al. (1992) or Lathrop et al. (1987). They predicted local regions using a pattern-matching method.

2) *Stochastic rule learning.*
The relationship between test regions and the two states, i.e. α-helix or not, may be represented by a stochastic rule, which is a probability distribution assigning, to each test region in a sequence, a probability that it is in an α-helix.

Furthermore, it is assumed that an individual stochastic rule is represented as the product of a number of stochastic rules with finite partitioning.

3) *Optimal categorization of amino acids with respect to the MDL principle.*
The SR method allows stochastic rules to make use of any numerical attributes (e.g. physicochemical characteristics) of amino acids in place of their symbols. Although a number of previously devised methods exist, in which amino acids are categorized on the basis of their numerical attributes (e.g. Taylor, 1986a; Nagano, 1977), such categorization is not theoretically guaranteed to be optimal. The SR method categorizes amino acids for each residue position, on the basis of Rissanen's minimum description length (MDL) principle, for which its optimality (Rissanen, 1978, 1989) and rapid statistical convergence (Yamanishi, 1992) in probabilistic model estimation are guaranteed. In addition, we note that until we established the SR method, the MDL principle had never been widely applied to probabilistic model estimation, especially in the field of computational biology except for the problem of selecting motifs (Yamanishi & Konagaya, 1991), to the best of our knowledge.

4) *Use of a number of training and test sequences.*
We use a number of aligned sequences not only for learning stochastic rules for each α-helix but also for predicting α-helices in a given test sequence. The idea of using a multiple number of homologous sequences for training data was used by the SR method (Mamitsuka & Yamanishi, 1992) and also by Rost and Sander (1993a, 1993b) in a work independent of ours. They used sequence profiles in place of twenty discrete amino acid symbols to train neural networks. Furthermore, the idea of using test sequences homologous to a given test sequence had not been employed by any other method until we established our method, in a similar context, to the best of our knowledge.

In our experiments, we investigated how well our proposed method predicts α-helices in comparison with the neural network learning methods, most specifically, the QS method (essentially including Hayward et al.'s method as a special case) and the RS method because the RS was one of the methods (e.g. Rost & Sander, 1993a) which achieved the highest prediction accuracy when we proposed the SR method and the QS method was the original method from which the RS method was derived. Experimental results show that the SR method achieved an average prediction accuracy of 81% while the RS method achieved a prediction accuracy of 80 to 82% depending on its initial values. This result indicates that the SR method reached the same level as that of the RS method, i.e. the highest level when we proposed the SR method.

Finally, we briefly explain the progress that has been made in the area of problems in predicting secondary structures, since we proposed our method. No new method for the two-state prediction problem has been proposed, to the best of out knowledge. On the other hand, approaches for the three-state prediction problem have been proposed, and they have slightly improved the prediction accuracy of the problem to the point where it is no approximately 75% (Frishman & Argos, 1997;

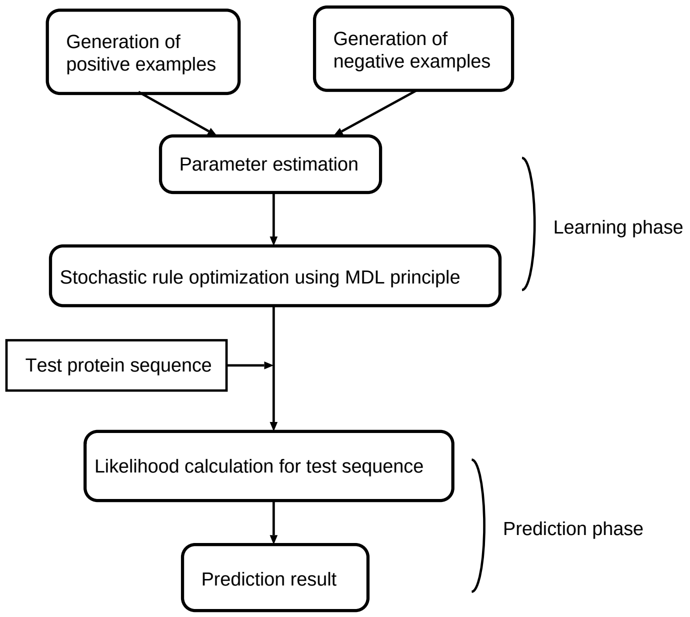

Generation of positive examples
Generation of negative examples
Parameter estimation
Stochastic rule optimization using MDL principle
Learning phase
Test protein sequence
Likelihood calculation for test sequence
Prediction result
Prediction phase

Figure 3.1: Outline of the SR method

Cuff & Barton, 1999). Thus, if the two-state prediction is performed by the approaches proposed recently, the prediction accuracy obtained by the SR method may be slightly improved. However, the number of presently available sequences or protein structures is far larger than that obtained when we proposed the SR method, and if all the data which is currently available were incorporated into the SR method, the accuracy of the method described in this chapter would be improved. Therefore, we believe that the SR method is one of the methods with the highest prediction accuracy in the field of the two-state prediction problem even at present.

3.2 Stochastic Rule Learning Method

Our method (the SR method) has three phases: example-generation, learning, and prediction, as shown in Figure 3.1. Example generation further consists of positive and negative example generation.

3.2.1 Example Generation

As a general rule, generating examples might be written in the 'Experimental Results' section, but our method has its own highly important feature in generating examples, and thus we here describe the method to generate training and test examples. Proteins with known three-dimensional structures and assigned α-helices were obtained from the database of Homology-derived Structures and Sequence alignments of Proteins (HSSP) Ver 1.0 (Sander & Schneider, 1991) for use both as training examples and test examples.

Training Example

We chose 25 proteins from the HSSP Ver 1.0 for use as training examples. We selected these to meet the following three separate conditions: 1) at least 50 additional sequences of each of the 25 proteins must be available from the HSSP database, 2) each of the 50 or more sequences obtained must possess at least 30% homology to its corresponding protein with known structure, and 3) each of the 25 proteins must belong to the PDB_SELECT 25% list developed by Hoboem et al. (Hoboem, Scharf, Schneider, & Sander, 1992), in which all the proteins selected possess no more than 25% pairwise sequence homology to any proteins in the PDB. The 25 proteins selected are shown in Table 3.1. In

PDB code	Protein	# of residues	# of α-helices	# of sequences from the HSSP database
1atn_A	Deoxyribonuclease I	372	10	97 ~ 100
1ccr	Cytochrome c	111	3	108 ~ 110
1clm	Calmodulin	144	7	138 ~ 147
1cpc_L	C-phycocyanin	172	8	55 ~ 56
1etu	Elongation factor tu	177	4	62 ~ 73
1fxi_A	Ferredoxin I	96	1	71
1grd_A	Glucocorticoid receptor	81	2	82
1hdd_C	Engrailed homeodomain	57	3	206 ~ 208
1hge_B	Hemagglutinin	175	4	97
1hsb_A	Human class I histocompatibility antigen	270	4	154 ~ 156
1izb_B	Insulin mutant	30	1	71
1lz3	Lysozyme	129	3	54 ~ 55
1mbd	Myoglobin	153	6	77
1pbx_A	Hemoglobin	142	6	480 ~ 482
1ppf_E	Leukocyte elastase	218	1	121
1ppn	Papain	212	4	50 ~ 51
1rnd	Ribonuclease A	124	2	57
1sgt	Trypsin	223	2	103 ~ 106
1sha_A	Tyrosin kinase transforming protein	103	2	70
3rub_S	Ribulose 1,5-bisphosphate carboxylase/oxylase	123	2	84 ~ 86
3sgb_I	Proteinase B	50	1	75
4bp2	Phospholipase A-2	117	3	83 ~ 106
4gpd_1	Aspartate dehydrogenase	333	6	66 ~ 79
5p21	C-H-ras H21 protein	166	5	115 ~ 116
9rub_B	Ribulose 1,5-biphosphate	458	13	81 ~ 92

Table 3.1: 25 training proteins

the table, codes in the PDB_SELECT 25% list are shown at the left side of the table. For each of the proteins, its name, the number of residues, the number of α-helix regions selected, and the number of the sequences selected are shown. Total number of residues is 4235 with 34 % residues in α-helix regions.

Generation of Positive Examples The sub-sequences in the HSSP sequences which corresponded to α-helices in the 25 proteins were themselves considered to be α-helices, and we call each of these a *positive example* of its corresponding α-helix. Alignment in the HSSP database was perfect across all these α-helices in the sense that no empty residue spaces were observed among them.

Generation of Negative Examples For each of the α-helices in the 25 proteins for positive examples, we randomly selected from the HSSP database *negative examples*, each of which is in a non-α-helix and has the same number of residues as the α-helix for which it is being selected.

Test Example

We selected 30 proteins as test proteins which would meet the following four conditions: 1) each of the 30 proteins must belong to the PDB_SELECT 25% list so that it is not homologous in its primary structure to any of the individual training proteins, 2) at least three additional examples of each of the 30 proteins must be available from the HSSP database, 3) at least one of the additional examples obtained must possess from 30 to 70% homology to its corresponding PDB_SELECT 25% list-supplied protein, 4) none of the 30 proteins must belong to the family into which each of the training proteins is classified, so that none of the additional examples are homologous to any of positive examples.

Each of the additional examples from the HSSP database is considered to be an aligned test example for its corresponding one of the 30 test proteins. Selected test proteins are shown in Table 3.2. In the table, codes in the PDB_SELECT 25% list are shown at the left side of the table. To each

PDB code	Protein	# of residues	# of sequences from the HSSP database
1caj	Carbonic anhydrase	258	23
1cdt_A	Cardiotoxin	60	93
1fc1_A	Immunoglobulin Fc fragment	206	57
1fha	Ferritin mutant	170	27
1gky	Guanylate kinase	186	9
1gmf_A	Macrophage colony stimulating factor	119	4
1gst_A	Glutahione s-transferase	217	24
1hil_A	IgG 2A Fab fragmaent	217	244
1hlh_A	Helix-loop-helix domain	64	26
1lig	Salmonella typhimurium	149	8
1lpe	Apolipoprotein	144	7
1mam_H	Antigen-binding fragment	217	203
1nxb	Neurotoxin	62	87
1pgd	Phosphogluconate dehydrogenase	469	7
1prc_M	Photosynthetic reaction center	323	7
1spa	Aspartate aminotransferase	396	20
1tie	Trypsin inhibitor	166	35
1tro_A	Trp repressor operator complex	104	3
1utg	Uteroglobin	70	4
2cts	Citrate synthetase	437	6
2msb_A	Mennnose-binding protein	111	18
2sn3	Neurotoxin variant-3	65	32
3chy	Signal transduction protein	128	26
3il8	Interleukin-8	68	23
3sod_O	Superoxide dismutase	151	48
4gcr	γ-B crystalin	185	53
4rxn	Rubredoxin	54	18
5hir	Hirudin	49	16
7api_B	Antitrypsin	36	18
7xia	D-xylose isomerase	387	10

Table 3.2: 30 test proteins

of the proteins, its name, the number of residues, the number of sequences derived from the HSSP database are shown. Total number of residues is 5268 with 38% residues in α-helix regions.

3.2.2 Learning Phase

Probabilistic Structure of Protein Sequences

Let us first describe the basic probabilistic structure.

Definition 3.1 (Basic Probabilistic Structure) (Mamitsuka & Yamanishi, 1995) Let $\mathcal{X}$ be a countable set, and n be a positive integer. Let $X_1, \cdots, X_n$ be random variables on $\mathcal{X}$. Let $\mathcal{Y}$ be a finite set. Let Y be a random variable on $\mathcal{Y}$. We call an element $y \in \mathcal{Y}$ a *label*. Let $\mathcal{D} = \mathcal{X}^n$. An element $x = x_1 \cdots x_n$ in $\mathcal{D}$ is called an *example*. We define a probability on $\mathcal{D} \times \mathcal{Y}$ by

$$P(Y = y \mid x) = \prod_{i=1}^{n} P(Y = y \mid X_i = x_i). \tag{3.1}$$

When this probability on $\mathcal{D} \times \mathcal{Y}$ is defined, we say that $\mathcal{D} \times \mathcal{Y}$ has a *basic probabilistic structure.* □

Hereafter, we assume that $\mathcal{D} \times \mathcal{Y}$ has a basic probabilistic structure.

We assume that this basic probabilistic structure is taken by any amino acid sequence. In the case of an amino acid sequence, each $x_i \in \mathcal{X}$ is an amino acid and an example $x \in \mathcal{D}$ is an amino acid sequence. This definition indicates that any amino acid may be uniquely identified as an element within $\mathcal{X}$. For example, if $\mathcal{X}$ is one-dimensional and all twenty amino acid symbols are represented along a line, then an amino acid is uniquely identified as an element along the line. If, on the other hand, $\mathcal{X}$ is two-dimensional, $\mathcal{X} = \mathcal{X}_a \times \mathcal{X}_b$ where $\mathcal{X}_a$ denotes hydrophobicity, and $\mathcal{X}_b$ denotes molecular weight, any amino acid is expressed as an element determined simply by its hydrophobicity and molecular weight combination. Furthermore, any amino acid sequence of length n can be uniquely identified as an element within $\mathcal{D}$.

Furthermore, for each x_i of $x = x_1 \cdots x_n$, we assume that $P(Y = y \mid X_i = x_i)$ is expressed as a stochastic rule with finite partitioning, which is given in Section 2.2.2. We recall the definition of a stochastic rule with finite partitioning.

Definition 3.2 (Stochastic Rule with Finite Partitioning) (Mamitsuka & Yamanishi, 1995) Let $\mathcal{X}^n \times \mathcal{Y}$ be a basic probabilistic structure. For each $i = 1, \cdots, n$, let $\{C_i^{(j)}\}_{1 \le j \le m_i}$ be a finite partitioning of $\mathcal{X}_i$, where each $C_i^{(j)}$ is called a *cell*. Let $p_i^{(j)}(y) \in [0, 1]$ $(y \in \mathcal{Y}, j = 1, \cdots, m_i)$ be a real-valued *probability parameter*, where $\sum_{y \in \mathcal{Y}} p_i^{(j)}(y) = 1$ is assumed for each i and each j. For each $i = 1, \cdots, n$, $P(Y = y \mid X_i = x_i)$ is given as a *stochastic rule with finite partitioning* as follows:

```
**For** $j := 1$ to $m_i$
  **if** $x_i \in C_i^{(j)}$ and $Y = y$ **then**
    $P(Y = y \mid X_i = x_i) = p_i^{(j)}(y)$.
```

□

Let $\theta_i(y)$ be an m_i-dimensional *probability parameter vector* which is given as follows:

$$\theta_i(y) \quad = \quad (p_i^{(1)}(y), \cdots, p_i^{(m_i)}(y)) \in [0, 1]^{m_i}$$

We write $P(Y = y \mid X_i = x_i)$ over $\theta_i(y)$ as $P(Y = y \mid X_i = x_i : \theta_i(y))$.

Figure 3.2 shows an example of stochastic rules with finite partitioning over $\mathcal{X} = \mathcal{X}_a \times \mathcal{X}_b$ where $\mathcal{X}_a$ denotes the range of hydrophobicity values, and $\mathcal{X}_b$ denotes the range of molecular weights where the total number of cells is nine. Notice that the number of cells does not exceed twenty because the number of possible different types of amino acids is at most twenty.

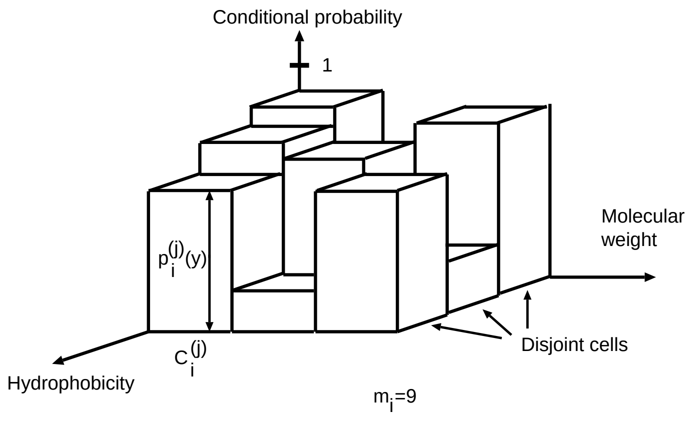


Figure 3.2: An example of stochastic rules with finite partitioning

Parameter Estimation

We use the Laplace estimator (Schreiber, 1985) to estimate the probability parameter vector $\theta_i(y)$ from given examples, and we denote it as $\hat{\theta}_i(y) = (\hat{p}_i^{(1)}(y), \cdots, \hat{p}_i^{(m_i)}(y))$ $(y \in \mathcal{Y}, i = 1, \cdots, n)$.

Definition 3.3 (Laplace Estimator) (Schreiber, 1985) Let $\mathcal{X}^n \times \mathcal{Y}$ be a basic probabilistic structure. For each $i = 1, \cdots, n$, let $\{C_i^{(j)}\}_{1 \le j \le m_i}$ be a finite partitioning of $\mathcal{X}_i$, and each $C_i^{(j)}$ is called a cell. Let $\mid \mathcal{Y} \mid$ be the number of labels.

When a set of sequences is given, for each $i = 1, \cdots, n$ and each $y \in \mathcal{Y}$, let $N_i^{(j)}(y)$ be the number of instances which are in a cell $C_i^{(j)}$. Let $N_i^{(j)} = \sum_{y \in \mathcal{Y}} N_i^{(j)}(y)$. For each $i = 1, \cdots, n$, the *Laplace estimator* $\hat{p}_i^{(j)}(y)$ is given as follows:

$$\hat{p}_i^{(j)}(y) = \frac{N_i^{(j)}(y) + 1}{N_i^{(j)} + \mid \mathcal{Y} \mid} \qquad (j = 1, \cdots, m_i). \tag{3.2}$$

□

It is known that the Laplace estimator can be derived from the Bayesian assumption (Schreiber, 1985). This is of practical use in our estimation problem because it allows us to avoid a situation in which the estimator is 1 or 0. Also notice that $\hat{p}_i^{(j)} = \frac{1}{|\mathcal{Y}|}$ when $N_i^{(j)} = 0$.

In the two-state prediction problem, we consider only two classes, i.e. α-helix or not. Thus, $\mid \mathcal{Y} \mid = 2$. For $y \in \mathcal{Y}$, we can write Eq. (3.2) as follows:

$$\hat{p}_i^{(j)}(y) = \frac{N_i^{(j)}(y) + 1}{N_i^{(j)} + 2} \qquad (j = 1, \cdots, m_i).$$

Optimization of Stochastic Rules

In trying to construct a stochastic rule which has high predictive performance, a major problem lies in how to determine a finite partitioning of a domain $\mathcal{X}$. If a finite partitioning is too complicated (i.e. the number of cells is too large), the rule may be expected to be affected by statistical irregularities in given examples (i.e. overfitting problem), even though it fits the given examples well. This sort of rule, then, cannot be relied upon to predict α-helices in test examples well. On the other hand, if a finite partitioning is too simple (i.e. the number of the cells is too small), there is a strong likelihood that the rule will not learn enough from given examples, and it will be unlikely to perform well with test examples. In an attempt to determine optimal finite partitioning, so as to avoid both of these disadvantageous situations, we determine the optimal number of cells, and the optimal placement of partitions.

We establish a new algorithm based on the minimum description length (MDL) principle (Rissanen, 1978, 1989) as a criterion to evaluate the optimality of any given structure of a stochastic rule.

As mentioned in Section 2.2.2, the MDL principle says that the optimal finite partitioning is that which minimizes the following two types of description length:

$$\text{(description length for instances relative to a given rule)} \\ +\text{(description length for the rule itself)}.$$

Once a stochastic rule with finite partitioning is constructed, we can calculate the description length relative to the rule.

The description length for given instances is given as follows:

$$-\sum_{1\le j\le m_i}\sum_{y\in\mathcal{Y}} N_i^{(j)}(y)\log \hat{p}_i^{(j)}(y)$$

The description length for the rule itself can be reduced to the description length of all $\hat{p}_i^{(j)}(y)$. Notice here $\frac{\log(N_i^{(j)})}{2}$ bits are needed to describe each $\hat{p}_i^{(j)}(y)$, since $\hat{p}_i^{(j)}(y)$ can be estimated with accuracy of $O(\frac{1}{\sqrt{N_i^{(j)}}})$ $(j = 1, \cdots, m_i)$ (see Rissanen, 1989; Yamanishi, 1992). Thus the description length for the rule itself is calculated as follows:

$$\sum_{1\le j\le m_i} \frac{\log N_i^{(j)}}{2}.$$

Definition 3.4 (Description Length of Stochastic Rule with Finite Partitioning) (Mamitsuka & Yamanishi, 1995) Let $\mathcal{X}^n \times \mathcal{Y}$ be a basic probabilistic structure. For each $i = 1, \cdots, n$, let $\{C_i^{(j)}\}_{1\le j\le m_i}$ be a finite partitioning of $\mathcal{X}_i$, and each $C_i^{(j)}$ is called a cell. For each $i = 1, \cdots, n, j = 1, \cdots, m_i, y \in \mathcal{Y}$, let $\hat{p}_i^{(j)}(y)$ be the Laplace estimator. When a set of sequences is given, for each $i = 1, \cdots, n$ and each $y \in \mathcal{Y}$, let $N_i^{(j)}(y)$ be the number of instances which are in a cell $C_i^{(j)}$. Let $N_i^{(j)} = \sum_{y\in\mathcal{Y}} N_i^{(j)}(y)$.

For each $i = 1, \cdots, n$, the description length of stochastic rule with finite partitioning is given as follows:

$$-\sum_{1\le j\le m_i}\sum_{y\in\mathcal{Y}} N_i^{(j)}(y)\log \hat{p}_i^{(j)}(y) + \sum_{1\le j\le m_i} \frac{\log N_i^{(j)}}{2}. \tag{3.3}$$

□

The optimal number of cells can be estimated as that which minimizes Eq. (3.3), according to the MDL principle given by Eq. (2.4).

In the two-state prediction problem, as the number of classes is two, let $\mathcal{Y} = \{0, 1\}$ for simplicity. For the problem, the description length of a stochastic rule with finite partitioning is as follows:

$$- \sum_{1 \leq j \leq m_i} \{N_i^{(j)}(0) \log \hat{p}_i^{(j)}(0) + N_i^{(j)}(1) \log \hat{p}_i^{(j)}(1)\} + \sum_{1 \leq j \leq m_i} \frac{\log N_i^{(j)}}{2}. \tag{3.4}$$

Once the optimal number of cells is determined by the MDL principle, we can obtain the probability parameters of a stochastic rule with finite partitioning for each $\mathcal{X} \times \mathcal{Y}$. The overall learning algorithm of the SR method is given as follows:

Algorithm 3.5 (Learning Stochastic Rule with Finite Partitioning)
input: examples with their labels
output: stochastic rule with finite partitioning, optimized in terms of the MDL principle
1: With varying the size of a cell, repeat a finite partitioning on $\mathcal{X}$ to generate a set of cells, each of which has an equal size. 2: For each set of cells on $\mathcal{X}$, calculate the description length according to Eq. (3.4), and retain the set whose description length is the smallest among all the sets tested
3: Calculate the Laplace estimators attached to the cells of the set retained □

3.2.3 Prediction Phase

Likelihood Calculation

We explain the way to calculate the likelihood that each part of a test sequence has a label. We assume that stochastic rules with finite partitioning were constructed. Using the stochastic rules, we calculate the likelihood that a given part of a test sequence has a label, more concretely, an α-helix.

Let $\mathcal{X}^n \times \mathcal{Y}$ be a basic probabilistic structure. For each $i = 1, \cdots, n$, let $\{C_i^{(j)}\}_{1 \leq j \leq m_i}$ be a finite partitioning of $\mathcal{X}$, where each $C_i^{(j)}$ is called a cell. For each label $y \in \mathcal{Y}$ and each $i = 1, \cdots, n$, $\hat{\theta}_i(y) = (\hat{p}_i^{(1)}(y), \cdots, \hat{p}_i^{(m_i)}(y))$ be a vector of the Laplace estimators.

For an integer $t < n$, let $\mathcal{X}^t \times \mathcal{Y}$ be a basic probabilistic structure, and we call $\mathcal{R} = \mathcal{X}^t$ a *matching region*. For $\mathcal{D} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$, let $\mathcal{R}_1, \cdots, \mathcal{R}_{n-t+1}$ be $n - t + 1$ matching regions obtained from $\mathcal{D}$, where $\mathcal{R}_i = \mathcal{X}_i \times \cdots \times \mathcal{X}_{i+t-1}$ $(i = 1, \cdots, n-t+1)$. For each label $y \in \mathcal{Y}$ and each $k = 1, \cdots, n-t+1$, let $\hat{\omega}_k(y)$ be the $(\sum_{i=1}^{t} m_{k+i-1})$-dimensional probability parameter vector for $\mathcal{R}_k$, which is given as follows:

$$\hat{\omega}_k(y) = (\hat{\theta}_k(y), \cdots, \hat{\theta}_{k+t-1}(y)).$$

An element $v = x_1 \cdots x_t \in \mathcal{R}$ is called a *test region*. For a label y, a test region v and a matching region $\mathcal{R}_k$, let $P(Y = y \mid v : \hat{\omega}_k(y))$ be the *likelihood* that $Y = y$ given v over $\hat{\omega}_k(y)$, which is as follows:

$$P(Y = y \mid v : \hat{\omega}_k(y)) = \prod_{i=1}^{t} P(Y = y \mid X_i = x_i : \hat{\theta}_{k+i-1}(y)).$$

For a label y, a test region v and $\mathcal{D}$, let $P(Y = y \mid v)$ be the likelihood that $Y = y$ given v, which is as follows:

$$P(Y = y \mid v) \quad = \quad \max_{1 \leq k \leq n-t+1} P(Y = y \mid v : \hat{\omega}_k(y)).$$

In the two-state prediction problem for amino acid sequences, a test region is a substring of a test amino acid sequence, and a matching region corresponds to a part of a given α-helix. Furthermore,

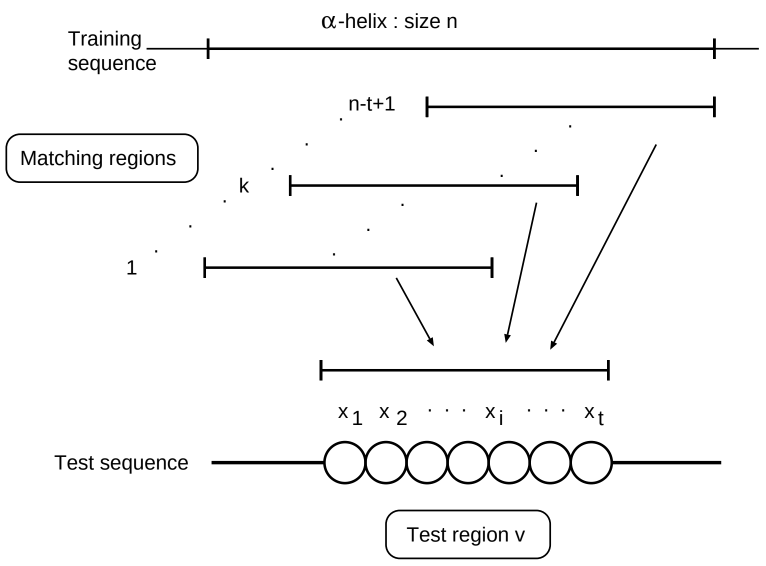


Figure 3.3: A schematic model of likelihood calculation

$\mathcal{Y} = \{0, 1\}$, and we let $Y = 1$ if a given example is an α-helix. Then, $P(Y = 1|v)$ is regarded as the likelihood that a test region v is in an α-helix. Figure 3.3 shows a schematic model of calculating the likelihood that a test region v is in a given α-helix.

Up to this point, we have assumed that there is only a single α-helix. In an actual amino acid sequence, however, there exist a number of α-helices, each of which can be used for training. Furthermore, there are a variety of amino acid sequences containing α-helices. Let $P^{(i)}(Y = 1 \mid v)$ be the likelihood that a test region v is in an α-helix i. If there are s α-helices, for a test example v, let $\tilde{P}(Y = 1 \mid v)$ be the likelihood that v is in one of given s α-helices, which is as follows:

$$\tilde{P}(Y = 1 \mid v) \quad = \quad \max_{1 \le i \le s} P^{(i)}(Y = 1 \mid v).$$

When $\tilde{P}(Y = 1 \mid v) = P^{(j)}(Y = 1 \mid v)$, and the value of $\tilde{P}(Y = 1 \mid v)$ is relatively high, we can guess that v is an α-helix whose probabilistic feature is similar to α-helix j. Henceforth, we write $P(Y = 1 \mid v)$ for $\tilde{P}(Y = 1 \mid v)$.

Prediction Curve

When a test sequence $u = x_1 \cdots x_l$ is given, we can obtain $l-t+1$ test regions. For an instance x_i of u, let $v_1(x_i), \cdots, v_t(x_i)$ be the t test regions, each of which includes x_i. Note that for instances near the ends of a test sequence, the number of test regions obtained is smaller than t. For each $i = 1, \cdots, l$, a *prediction curve* for α-helices is a mapping of $f : \mathcal{X} \to [0, 1]$ given by the following form:

$$f(x_i) \quad = \quad \max_{1 \le j \le t} P(Y = 1 \mid v_j(x_i)).$$

Figure 3.4 shows a schematic model of calculating a prediction curve for a given test sequence.

When a set of sequences $\mathcal{U} = \{u_1, \cdots, u_W\}$ is given, let $f_1(x_i), \cdots, f_W(x_i)$ $(i = 1, \cdots, l)$ be prediction curves of sequences $u_1, \cdots, u_W$, respectively. For $i = 1, \cdots, l$, a *prediction curve for aligned*

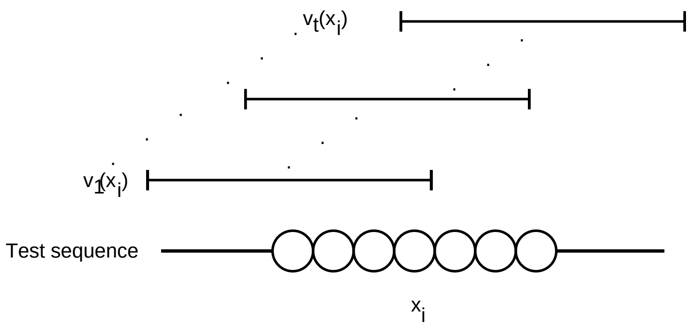


Figure 3.4: Test regions including x_i

sequences is a mapping of $g : \mathcal{X} \rightarrow [0,1]$ given by the following form:

$$g(x_i) \quad = \quad \min_{1 \leq j \leq W} f_j(x_i).$$

In our experiments, when a test sequence is given, there is a case in which not only the given sequence but also sequences homologous to it are used. Thus, in this case, we regard the prediction curve for aligned sequences as a prediction curve for a given sequence.

3.3 Experimental Results

3.3.1 Performance Measure

In order to evaluate the SR method, we fix some threshold for a prediction curve $g(x_i)$. For an instance x_i in a given sequence, if $g(x_i)$ exceeds the threshold, then we predict that x_i is in an α-helix, otherwise we predicts that it is not. We evaluate the predictive performance of the SR method in a conventional residue-based way, to compare the SR method with neural network learning methods, though the SR method predicts whether a given test region is α-helix or not.

Let N be the number of all amino acids in a test sequence. Let N_{ca} be the number of correctly predicted amino acids in α-helices, and N_{cn} be the number of correctly predicted amino acids in non α-helices. Let N_{wa} be the number of wrongly predicted amino acids in α-helices and N_{wn} be the number of wrongly predicted amino acids in non-α-helices.

First, the *prediction accuracy* Q, which is the most popular measure in the protein secondary structure prediction problem, is given as follows:

$$Q = \frac{N_{ca} + N_{cn}}{N}. \tag{3.5}$$

Then, Matthews' correlation coefficient C (Mattews, 1975), which is also an popular alternative performance measure, is given as follows:

$$C = \frac{N_{ca} \times N_{cn} - N_{wa} \times N_{wn}}{(N_{ca} + N_{wa}) \times (N_{ca} + N_{wn}) \times (N_{cn} + N_{wa}) \times (N_{cn} + N_{wn})}.$$

3.3.2 Stochastic Rule Learning Method

Two types of strategies for the SR method

In our experiments, we used two versions of the SR method: s_0 and s_{MDL}.

s_0 denotes a prediction strategy in which $\mathcal{X} = \mathcal{X}_A$, which is one-dimensional and represents along its line the set of all twenty amino acid symbols. Thus, $\mid \mathcal{X}_A \mid = 20$. In s_0, the number of cells specifying a stochastic rule at each residue position is fixed at twenty, and the structure of the stochastic rule is not optimized.

s_{MDL} denotes a strategy in which $\mathcal{X} = \mathcal{X}_B = \mathcal{X}_a \times \mathcal{X}_b$, where $\mathcal{X}_a$ denotes the range of hydrophobicity, and $\mathcal{X}_b$ denotes the range of molecular weight. In s_{MDL}, the structure (i.e. the number of cells and their assignment) of a stochastic rule for each residue position is optimized based on the MDL principle. Here we used hydrophobicity values determined by Fauchere and Pliska (1983). Figure 3.5 shows an actual example of $\mathcal{X}_B$, in which all amino acids belong to a single cell.

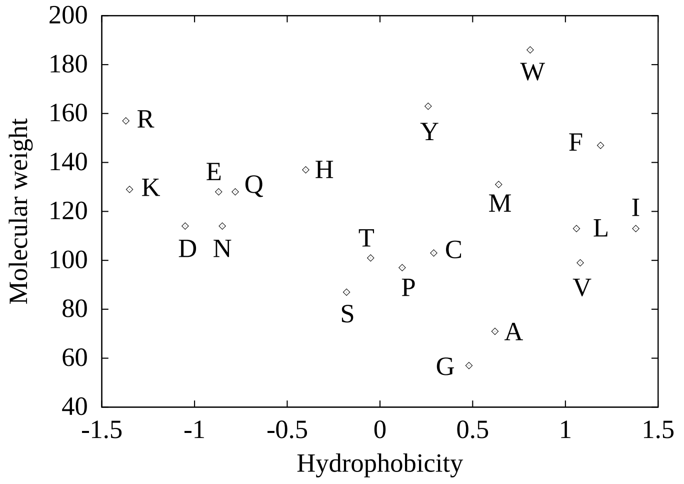


Figure 3.5: Domain $\mathcal{X}_B$

Size of Test Region

In the prediction phase, we fixed the size of a test region to be seven, since seven amino acids yield approximately two turns of an α-helix. Although eight is also a potential candidate for the size, we observed in preliminary experiments that seven gave better prediction accuracy.

Prediction Accuracy for Training Proteins

The prediction accuracy Q of the SR method depends on the threshold used as the cut-off line for predicting α-helices. The best average prediction accuracy of s_0 and s_{MDL} for 25 training proteins was 84%, where the threshold was 0.016 or 0.017.

Prediction Curve and Prediction Accuracy for Test Proteins

Figure 3.6 shows the prediction curve of 2msb_A by s_{MDL}. In the figure, a real line and a dotted

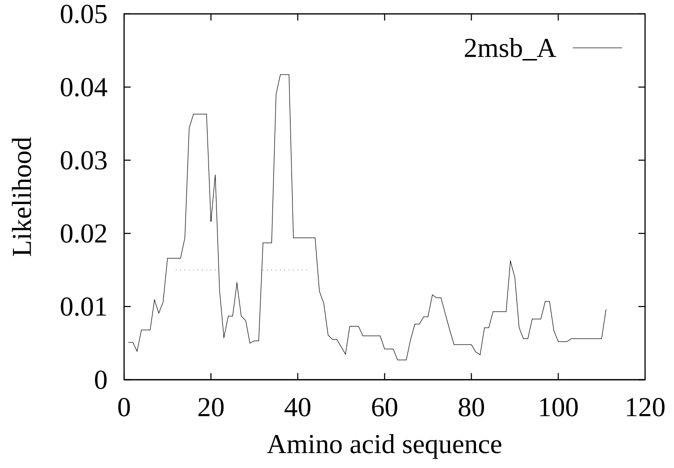


Figure 3.6: Prediction curve for 2msb_A

line respectively show a prediction curve and true α-helices of 2msb_A. The figure shows that the α-helices of 2msb_A were properly predicted and only a small part of the non-α-helix regions were overestimated as α-helix.

Figure 3.7 shows the average prediction accuracy for all the proteins given in Table 3.1. For s_0 and s_{MDL}, the best average prediction accuracy, 81%, was obtained by s_{MDL} when the threshold was 0.015. Table 3.3 shows the prediction accuracy of each test protein at this threshold. In the table, prediction accuracies for 30 test proteins are shown for the case when the threshold is 0.015. Codes in the PDB_SELECT 25% list are shown at the left side. To each of 30 proteins, the prediction accuracy (Q) and Mattew's correlation coefficient (C) of two strategies are shown. This result tells us that the optimization of stochastic rules using the MDL principle was effective to achieve the high prediction accuracy of 81%.

3.3.3 Neural Network Learning Method

We compared the predictive performance of the SR method with two neural network learning methods, i.e. the QS and RS methods, in terms of average prediction accuracies.

First, we briefly describe the outline of the QS and RS methods which we implemented in our experiments to apply them to the two-state prediction problem. Note that the RS method implemented here is not completely the same as those used in Rost and Sander (1993a, 1993b), since the implemented method did not use additional knowledge including secondary structure contents or filtering unrealistic predictions, which were employed in Rost and Sander (1993a, 1993b).

The QS Method

Training and Test Examples The 25 proteins shown in Table 3.1 were used as training examples, but positive and negative examples generated for the SR method were not used.

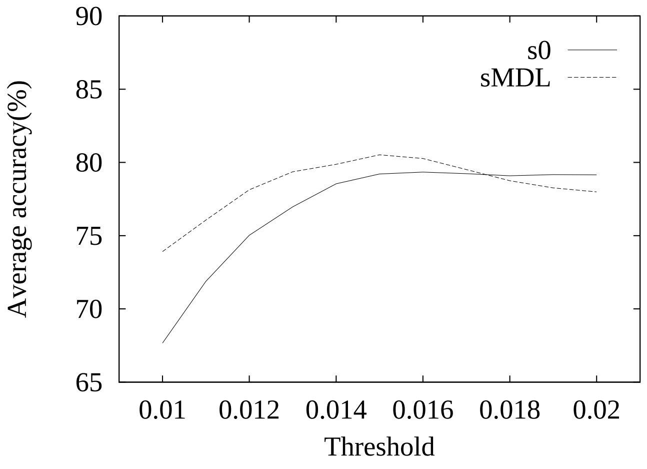


Figure 3.7: Average prediction accuracies for test proteins

The 30 proteins shown in Table 3.2 were used as test proteins, but the sequences aligned to them were not used in the test.

Network design The network used here is of the feed-forward type consisting of three layers, i.e. input, hidden, and output, and each layer has a number of *units*.

The input layer consists of a number of input *groups*, each of which further consists of 21 units. For each group, each of twenty units among the twenty-one corresponds to a single amino acid, and the remaining unit is a dummy one. For any given protein sequence, each amino acid position in the sequence is assigned sequentially to a group. In each group, only the unit corresponding to the assigned amino acid outputs "1," and all other units output "0." When an assigned sequence portion contains no amino acid, however, only the dummy unit outputs "1," and all other units output "0" in the group.

The hidden layer also comprises a number of units hereafter referred to as *hidden units*, and each input layer unit is connected individually to every one of the hidden units. Each hidden unit is, in turn, connected to both of the output layer units. These connections are referred to as *edges*, and each edge has a real-valued modifiable *weight*.

The output layer consists of only two units referred to as output units, i.e. an α-helix unit and a non-α-helix unit, the same as designed by Bohr et al. (1988).

Learning Phase We set a *window* whose length is equal to the number of input groups. By sliding the window through a given protein sequence, we obtain a number of *partial regions*, each of which has the same length as the window. When a partial region is provided to the input layer, if its central residue is in an α-helix, the α-helix unit should output "1," and the non-α-helix unit should output "0"; if its central residue is not in an α-helix, the α-helix unit should output "0," and the non-α-helix unit should output "1."

Parameters of the neural network were trained by two algorithms: propagation and back-propagation. Output values of the units in the hidden and output layers were calculated using the propagation

PDB code	Q_{s_0}	C_{s_0}	$Q_{s_{MDL}}$	$C_{s_{MDL}}$
1caj	0.79	0.32	0.90	0.46
1cdt_A	0.93	0.0	1.00	0.0
1fc1_A	0.83	-0.09	0.92	-0.03
1fha	0.71	0.34	0.72	0.41
1gky	0.72	0.43	0.77	0.54
1gmf_A	0.69	0.43	0.71	0.42
1gst_A	0.67	0.35	0.70	0.41
1hil_A	0.89	0.17	0.90	-0.05
1hlh_A	0.89	0.65	0.88	0.66
1lig	0.79	0.25	0.77	0.28
1lpe	0.83	0.21	0.76	-0.01
1mam_H	0.96	0.00	0.99	0.00
1nxb	0.97	0.00	1.00	0.00
1pgd	0.76	0.53	0.76	0.51
1prc_M	0.71	0.41	0.70	0.40
1spa	0.76	0.52	0.74	0.48
1tie	0.88	0.00	0.99	0.00
1tro_A	0.77	0.35	0.76	0.29
1utg	0.77	0.37	0.76	0.32
2cts	0.74	0.46	0.69	0.36
2msb_A	0.91	0.78	0.95	0.87
2sn3	0.86	-0.05	0.86	-0.05
3chy	0.83	0.67	0.77	0.58
3il8	0.79	0.52	0.87	0.69
3sod_O	0.91	-0.04	0.90	-0.05
4gcr	0.86	0.30	0.89	-0.05
4rxn	0.96	0.00	1.00	0.00
5hir	0.78	0.00	0.96	0.00
7api_B	0.92	0.00	1.00	0.00
7xia	0.72	0.45	0.71	0.41
total	0.79	0.57	0.81	0.59

Table 3.3: Prediction accuracies for test examples

algorithm, and weights attached to edges were trained using the back-propagation algorithm (e.g. Rumelhart et al., 1986). These are common algorithms to train typical feed-forward type neural networks as used in the QS and RS.

Prediction Phase The network predicts whether the central residue of a given partial region in a test sequence is in an α-helix or not. If the output value of the α-helix output unit is larger than that of the non-α-helix one, the network predicts that the region is in an α-helix, otherwise it predicts that the region is not.

The RS Method

Training and Test Examples The 25 proteins shown in Table 3.1 and aligned sequences homologous to them were used as training examples. Similarly, the 30 proteins shown in Table 3.2 and aligned sequences homologous to them were used in the test.

System The RS method consists of three *levels*: two neural network levels and a final level. Parameters of the two neural network levels are trained using examples while the final third level merely calculates the sum of the outputs of independent multiple second-level neural networks in prediction.

The First-level The first-level network is similar to the neural network used in the QS except that aligned multiple sequences were used as input data in the level. From the sequences, we calculate the probability distribution of twenty types of amino acids for each residue position, and when the position is given to a group in the input layer, each input unit in the group outputs its corresponding probability.

The Second-level The second-level network is the same as the network in the QS except that each group in the input layer of the second-level has only two input units for receiving two output values of the first-level. Output values of the first-level are directly assigned to an input group in the second-level network, which group corresponds to the central residue of the region given to the first-level.

The Third-level In the third level, the arithmetic sum of the output of several independent second-level neural networks is calculated for the α-helix units and the non-α-helix units, individually.

Learning Phase The learning algorithm of the RS in the first- and second-level, is the same as that of the QS. We set a window having the same size as the number of input groups. By sliding the window through a given training protein sequence, we obtain a number of partial regions having the same length as the window.

In the first-level, when the partial region with aligned multiple sequences is provided to the input layer, if its central residue is in an α-helix, "1" is given to the α-helix unit as a teacher signal, and "0" is given to the non-α-helix unit; otherwise teacher signals are reversely given to the output units. Similarly, in the second-level, when a partial region having output values of the first-level is provided to the input layer, the teacher signals are given to the output layer, depending on whether the central residue of the given region is in an α-helix.

Prediction Phase The prediction phase is the same as that of the QS. If the output value of the α-helix unit is larger than the non-α-helix one, the network predicts that the given region is in an α-helix, otherwise the network predicts that the region is not.

Implementation of Neural Networks

For each of weights in the networks, we randomly chose an initial value in a range of -0.5 to 0.5.

In the QS, we let the number of hidden units be 20, 40, and 60 and let the number of input groups be 11, 13, 15, 17, and 19.

In the RS, we let the number of input groups in the first-level network be 7, 9, 11, 13, 15, and 17 while the number of hidden units in both the first- and second-level and the number of input groups in the second-level were fixed to be 40 and 17, respectively. We used five independent neural networks for calculating the sum of their output values in the third-level.

Prediction for Test Proteins

The QS Method The average prediction accuracy was a range of 72 to 75%. None of the neural networks of the QS method could achieve an average prediction accuracy of 81%, which the SR attained for the same test proteins. This result shows that the SR method can provide higher predictive performance for the two-state prediction problem than the QS method.

QS	RS2s	RS3s	RS2m	RS3m	SR
$72 \sim 75$	$72 \sim 76$	$76 \sim 78$	$77 \sim 81$	$80 \sim 82$	81

Table 3.4: Average prediction accuracies (%)

The RS Method We tested four versions of the RS method, i.e. RS2s, RS2m, RS3s and RS3m. The RS2s and RS2m did not use the third level of the RS method while the RS3s and RS3m used the overall level. The RS2m used aligned multiple sequences in the test, while the RS2s used only each single sequence of the 30 test proteins. Similarly, the RS3m used multiple sequences homologous to each test protein while the RS3s used 30 test sequences only. In training, all the four methods used aligned multiple sequences which are homologous to 25 training proteins. In Table 3.4, we summarize the results of the average prediction accuracies obtained by the four methods.

The average prediction accuracy of the RS2s was a range of 72 to 76%, which was almost the same as that of the QS and was approximately 5 to 10% lower than that of the RS2m. This result implies that the use of multiple sequences in prediction is more effective for improving prediction accuracies than the use of them in training. The average prediction accuracy of the RS3m achieved a range of 80 to 82%, the upper limit of which was slightly better than that of the SR method, while most of the average prediction accuracies obtained by the RS2m could not attain the best prediction accuracy reached with the SR method.

From these results, we conclude that the SR method achieved the high prediction accuracy of 81%. This is the same level as that obtained by the RS method, namely the best method for predicting α-helices when we proposed the SR method.

3.4 Conclusion

We established, on the basis of the theory of stochastic rule learning, a new learning method for predicting α-helices and show its effectiveness by comparing its predictive performance with those of the neural network learning methods, especially the QS and RS methods. Specifically, we show that the SR method achieved 81% average prediction accuracy for 30 test proteins which consist of a wide variety of α-helix contents and each of which has not more than 25% pairwise sequence similarity to each of the training and other test proteins. We can conclude that when we proposed the SR method, it was one of the methods which provided the highest predictive performance in the two-state prediction problem.

The high predictive performance of the SR method can be attributed to the four characteristics described in the Introduction of this chapter. It should be noted that, in particular, the MDL principle plays an essential role in constructing optimal rules that have high prediction accuracy. As seen in Figure 3.7, s_{MDL}, in which the set of amino acids is categorized for each residue position using the MDL principle, actually gave approximately 2 to 3% higher prediction accuracy than s_0. This suggests that the MDL principle is a powerful tool for modeling probabilistic structures of α-helices.

Finally, we emphasize that, even beyond its superior predictive performance, the SR method has other advantages as well over the neural network learning based methods.

1. *Low computational complexity in the learning phase.*
   Computation time required for calculating stochastic rules depends on the size of partitions of

a domain for a single residue position and the number of given training examples, but it is basically much less than that required for neural network learning, since the learning phase in the SR method does not require any iteration as used in the back-propagation algorithm. In our experiments, the computation time for stochastic rule learning was less than one tenth of that for neural network learning.

2. *Comprehensibility of physico-chemical properties of α-helix.*
   The SR method can tell us which α-helix in training proteins has the same type of physico-chemical properties as the predicted α-helix. We cannot expect neural network learning methods, such as the QS and RS methods, to provide this kind of analysis.

Chapter 4

Representing Inter-residue Relations with Probabilistic Networks

4.1 Introduction

Motifs are generally functionally crucial patterns hidden in amino acid sequences, and the most important theme related to motifs is to represent them in some form which can be searched for in new sequences with reasonable accuracy and speed. A motif has been represented simply by a sequence pattern as used in the PROSITE database (Hofmann et al., 1999), or by a profile which corresponds to probability distributions of amino acids (Lüthy et al., 1994; Gribskov et al., 1990). All these simple approaches for representing a motif provide us with only 1-dimensional information on the motif, whereas the motif is expected to include remote *inter-residue relations* because its function is affected by interactions between residues which are three-dimensionally adjacent to each other but may be distant on the one-dimensional level. If we could extract such inter-residue relations within a motif and at the same time automatically display the relations in some visible form, the result would be a great aid in elucidating mechanisms related to the motif. A hidden Markov model (HMM), which was proposed to represent a profile or a motif, attempts to deal with such an inter-residue relation as a probability (Krogh et al., 1994a; Baldi et al., 1994). That is, a transition probability between two states in an HMM represents the probabilistic relation between two adjacent residues. Although an HMM can be trained by sequences with a variety of lengths, the structure of an HMM used in general is fixed as a model, in which only transitions between adjacent states are allowed. Thus such an HMM does not represent any relations between distant residues, i.e. remote inter-residue relations.

Motivated largely by these drawbacks, we focused on non-adjacent inter-residue relations and established a new method for automatically constructing networks to represent such relations, within a very short period of time (Mamitsuka, 1993, 1995). Our method constructs, from a number of aligned sequences for a short sequence, a probabilistic network, in which each node corresponds to a residue in the sequence, and a directed arc between nodes represents a probabilistic relation between the corresponding residues, i.e. a probabilistic inter-residue relation. Furthermore, a finite partitioning is done over a domain corresponding to a node, and thus we call our network the *probabilistic network with finite partitionings*. For simplicity, hereafter we refer to it as a probabilistic network in this Introduction. The idea behind our method for detecting probabilistic inter-residue relations hidden in given sequences is very simple: Several positions in a given sequence are said to have an inter-residue relation if substitutions in all of those positions in the sequences aligned to the sequence co-occur while such substitutions are not found in other positions. We quantify the strength of the inter-residue relation as a *conditional probability* in the probabilistic network.

We established a time-efficient algorithm to learn these probabilistic inter-residue relations, based on the minimum description length (MDL) principle (Rissanen, 1978, 1989) and a *greedy* algorithm. The MDL principle gives us a criterion to obtain the optimal probabilistic network using given data, and the greedy algorithm enables the network to be obtained efficiently in terms of computation time. We emphasize that we had to devise our algorithm to employ them, because there exist exponential combinations of all possible network structures. Under the circumstance, we divide our whole probabilistic network into partial probabilistic networks, each of which has a node and its predecessors, from each of which a directed arc goes to the node. This division gives us two merits. By doing this, first, the description length of a whole network is the sum of the description lengths of all partial probabilistic networks. Second, based on this, if we obtain the optimal probabilistic network with respect to the MDL principle, we merely have to search, in each partial probabilistic network, for predecessors which reduce the description length of the partial network by the largest amount. However, even for a partial network, there exist exponential combinations of predecessors, and there exists a no-cycle constraint in the directed acyclic graph of our probabilistic network. That is, the next problem is how to search efficiently for predecessors for each partial network among all possible combinations satisfying the no-cycle constraint. Thus, we use a greedy algorithm, which for a whole network iteratively chooses a predecessor which reduces the description length by the largest amount among all possible candidates which do not violate the no-cycle constraint. It should be noted that the description length criterion we use can be calculated efficiently in terms of computation time. Cooper and Herskovits (1992) used a similar greedy algorithm for searching for the optimal structure of a probabilistic network, but their method requires a great deal of time (i.e. a day or more) to calculate the criterion to determine whether a network is being improved or not when a new arc is added to the network. In contrast, our description length criterion can be calculated in a much shorter time (i.e. within a second).

In our experiments, we focused on a loop region of the EF-hand motif, which is peculiar to calcium binding proteins (Moncrief et al., 1990; Nakayama et al., 1992), and obtained actual amino acid sequences of the region. Experimental results show that our method constructed a probabilistic network for the region using the given sequences within a single second, and the network captured several important three-dimensional features of the region. From these results, we conclude that our method can provide crucial information for analyzing protein sequences.

Before we proposed our method, two other groups also tried to apply this type of probabilistic networks to represent an amino acid sequence. Delcher et al. (1993) used a probabilistic network (tree) for a protein secondary structure prediction problem. They, however, used a network with a fixed structure that did not provide any information on distant residues. Klingler and Brutlag (1994) also used a probabilistic network for representing correlations between residue positions. They captured some characteristics of α-helices which are generally known, though their method deals with only correlations between residue pairs. We emphasize that our method directly deals with relations among more than two residues, and automatically constructs a near-optimal network with respect to the MDL principle.

Since we proposed our method, no probabilistic networks have been applied to represent biological sequences, to the best of our knowledge. This is because HMMs, which as mentioned earlier are well suited for capturing a profile in given sequences with variable lengths, have been proven to be useful for several problems in biological sequence analysis (Durbin et al., 1998) and HMMs are now generally used as stochastic knowledge representations in the computational biology field. However, probabilistic networks themselves are even now widely applied to represent uncertain knowledge in a variety of other fields, and thus methods for learning or modifying them are now still extensively studied in one field of artificial intelligence (Langley, Provan, & Smyth, 1997). Here, it is worth noting that one disadvantage of most of the methods is that they require a large amount of computation time to calculate the optimal probabilistic network, (e.g. Heckerman, Geiger, & Chickering, 1995). In contrast, as mentioned earlier, our method allows us to calculate a near-optimal network very efficiently in terms of computation time. From this viewpoint, our method is still useful for learning a near-optimal probabilistic network in a limited period of time.

4.2 Method

4.2.1 Probabilistic Network with Finite Partitionings

We recall the definition of probabilistic network with finite partitionings, which is given in Chapter 2.

Let $K = (\mathcal{S}, \mathcal{A})$ be a directed acyclic graph with the node set $\mathcal{S} = \{1, \cdots, n\}$ and the arc set $\mathcal{A}$. If $i \rightarrow j$ is an arc, we say that i is a *predecessor* of j. Let $\pi_i = \{\pi_i^{(1)}, \cdots, \pi_i^{(k_i)}\} \subset \mathcal{S}$ be the set of the predecessors of node i. Note that $i \notin \pi_i$.

Definition 4.1 (Probabilistic Network with Finite Partitionings) (Mamitsuka, 1995) A *probabilistic network with finite partitionings* $\mathcal{N}$ consists of the following:
(i) A directed acyclic graph $K = (\mathcal{S}, \mathcal{A})$.
(ii) Random variables X_i on domains $\mathcal{X}_i$ for $i = 1, \cdots, n$.
(iii) A finite partitioning $\{C_i^{(j)}\}_{1 \le j \le m_i}$ of $\mathcal{X}_i$ for each $i = 1, \cdots, n$. □

Let $\mathcal{V} = \{X_1, \cdots, X_n\}$.
For a variable X_i, we define the set of *predecessor variables* $\Pi_i = \{X_{\pi_i^{(1)}}, \cdots, X_{\pi_i^{(k_i)}}\} \subset \mathcal{V}$.

For $x_i \in \mathcal{X}_i$, let $e_i(x_i)$ be the cell number specified by x_i which is given as follows:

```
For  $i := 1$ to $n$
  For  $j := 1$ to $m_i$
   if $x_i \in C_i^{(j)}$ then
     $e_i(x_i) = j$.
```

Let $x = x_1 \cdots x_n$ be an example in the domain $\mathcal{D} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$.
Let $e_{\pi_i}(x_{\pi_i}) = (e_{\pi_i^{(1)}}(x_{\pi_i^{(1)}}), \cdots, e_{\pi_i^{(k_i)}}(x_{\pi_i^{(k_i)}}))$. Let $C_{\pi_i}^{(e_{\pi_i}(x_{\pi_i}))} = (C_{\pi_i^{(1)}}^{(e_{\pi_i^{(1)}}(x_{\pi_i^{(1)}}))}, \cdots, C_{\pi_i^{(k_i)}}^{(e_{\pi_i^{(k_i)}}(x_{\pi_i^{(k_i)}}))})$.
For a cell $C_i^{(e_i(x_i))}$ and $C_{\pi_i}^{(e_{\pi_i}(x_{\pi_i}))}$, the *conditional probability* $p_{e_i(x_i)|e_{\pi_i}(x_{\pi_i})}(i|\pi_i)$ is the probability that X_i is in $C_i^{(e_i(x_i))}$ given that $X_{\pi_i^{(j)}}$ is in $C_{\pi_i^{(j)}}^{(e_{\pi_i^{(j)}}(x_{\pi_i^{(j)}}))}$ for each $j = 1, \cdots, k_i$. For x, the *joint probability* $P(x)$ given by $\mathcal{N}$ is as follows:

$$P(x) = \prod_{1 \le i \le n} p_{e_i(x_i)|e_{\pi_i}(x_{\pi_i})}(i|\pi_i) \tag{4.1}$$

4.2.2 Efficient MDL Learning Algorithm for Probabilistic Networks

Optimal Finite Partitioning with the MDL Principle

The main purpose of a finite partitioning is to cut down on the computational requirement in calculating the description length of a probabilistic network with finite partitionings. Another possible advantage of a finite partitioning is to improve the predictive performance of a network, when the network is trained by a number of sequences belonging to a class and an unknown sequence is given to predict whether it belongs to the class or not. This advantage is common to the problem of predicting α-helices described in the previous chapter. The method for a finite partitioning follows the same process as the one described in Chapter 3.

As in the two-state prediction problem in Chapter 3, we consider only two labels, i.e. whether it is a motif or not. Thus, let $\mathcal{Y} = \{0, 1\}$, and $Y = 1$ if a given example is a motif.

For a domain $\mathcal{X}_i$, a label $y = 0, 1$ and $j = 1, \cdots, m_i$, let $N_i^{(j)}(y)$ be the number of instances which are in a cell $C_i^{(j)}$. Let $N_i^{(j)} = \sum_{y \in \mathcal{Y}} N_i^{(j)}(y)$. For each $i = 1, \cdots, n$, the Laplace estimator $\hat{p}_i^{(j)}(y)$ is given as follows:

$$\hat{p}_i^{(j)}(y) = \frac{N_i^{(j)}(y) + 1}{N_i^{(j)} + 2} \qquad (j = 1, \cdots, m_i, y = 0, 1).$$

The description length for a finite partitioning is given as follows:

$$-\sum_{1 \le j \le m_i} \log\{\hat{p}_i^{(j)}(1)^{N_i^{(j)}(1)}(1 - \hat{p}_i^{(j)}(1))^{N_i^{(j)} - N_i^{(j)}(1)}\} + \sum_{1 \le j \le m_i} \frac{\log N_i^{(j)}}{2}, \tag{4.2}$$

According to the MDL principle, the optimal partitioning is obtained when the description length is minimized. Accordingly, we choose the partitioning which has a minimal description length (4.2). The overall algorithm is given as follows:

Algorithm 4.2 (Obtaining Optimal Finite Partitioning) (Mamitsuka, 1995)
input: positive and negative examples
output: optimal finite partitioning
1: For each possible set of cells over the domain $\mathcal{X}_i$ ($i = 1, \cdots, n$), calculate its description length according to (4.2), and retain the set whose description length is the smallest among all the sets tested.
2: Output the set retained. □

MDL Learning for Probabilistic Network with Finite Partitionings

In this section, we consider a situation in which only examples of a motif, i.e. positive examples, are given. Note that this situation is different from the one in the previous section, in which both positive and negative examples are given.

We first divide a directed acyclic graph of the probabilistic network with finite partitionings into smaller graphs. For each $i = 1, \cdots, n$, let $K_i = (\mathcal{S}_i, \mathcal{A}_i)$ be a *partial graph* of a directed acyclic graph $K = (\mathcal{S}, \mathcal{A})$, with the node set $\mathcal{S}_i = \{i\} \cup \pi_i$ and the arc set $\mathcal{A}_i = \{\pi_i^{(1)} \to i, \cdots, \pi_i^{(k_i)} \to i\}$. Note that $\mathcal{A} = \cup_{1 \le i \le n} \mathcal{A}_i$. For each $i = 1, \cdots, n$, a *partial probabilistic network with finite partionings* $\mathcal{N}_i$ consists of a partial graph K_i, random variables X_t on domains $\mathcal{X}_t$ for each $t \in \mathcal{S}_i$, and finite partitionings $\{C_t^{(j)}\}_{1 \le j \le m_t}$ of $\mathcal{X}_t$ for each $t \in \mathcal{S}_i$.

Let Γ be the description length of $\mathcal{N}$, and let Γ_i be the description length of $\mathcal{N}_i$. Note that $\Gamma = \sum_{1 \le i \le n} \Gamma_i$.

The MDL principle says that the optimal probabilistic network has a minimal description length. Thus, we have to minimize the description length of each partial network with finite partitionings to obtain the optimal probabilistic network with finite partitionings. In other words, we simply choose the predecessors of a node i which reduce the description length of $\mathcal{N}_i$ by the largest amount. Below, we formulate the description length of $\mathcal{N}_i$.

Let $N_{j_1,\cdots,j_{k_i}}(\pi_i)$ be the number of examples in which $X_{\pi_i^{(t)}}$ is in $C^{(j_t)}_{\pi_i^{(t)}}$ for each $t = 1, \cdots, k_i$. Let $N_{j_0,j_1,\cdots,j_{k_i}}(i,\pi_i)$ be the number of examples in which X_i is in $C_i^{(j_0)}$ and $X_{\pi_i^{(t)}}$ is in $C^{(j_t)}_{\pi_i^{(t)}}$ for each $t = 1, \cdots, k_i$. Let $\hat{p}_{j_0|j_1,\cdots,j_{k_i}}(i|\pi_i)$ be the Laplace estimator (Schreiber, 1985) given as follows:

$$\hat{p}_{j_0|j_1,\cdots,j_{k_i}}(i|\pi_i) \;=\; \frac{N_{j_0,j_1,\cdots,j_{k(i)}}(i,\pi_i)+1}{N_{j_1,\cdots,j_{k(i)}}(\pi_i)+m_i}.$$

The description length consists of two parts, namely the description length for given examples and the description length for $\mathcal{N}_i$ itself.

First, for $\mathcal{N}_i$ $(i = 1, \cdots, n)$, the description length for given examples is given by the minus logarithm of the maximum likelihood as follows:

$$-\sum_{1\le j_0\le m_i}\sum_{1\le j_1\le m_{\pi_i^{(1)}}}\cdots\sum_{1\le j_{k_i}\le m_{\pi_i^{(k_i)}}} N_{j_0,j_1,\cdots,j_{k_i}}(i,\pi_i)\log\hat{p}_{j_0|j_1,\cdots,j_{k_i}}(i|\pi_i). \tag{4.3}$$

Then, the description length for $\mathcal{N}_i$ itself is given as follows :

$$\sum_{1\le j_0\le m_i}\sum_{1\le j_1\le m_{\pi_i^{(1)}}}\cdots\sum_{1\le j_{k(i)}\le m_{\pi_i^{(k_i)}}} \frac{\log N_{j_0,j_1,\cdots,j_{k_i}}(i,\pi_i)}{2}. \tag{4.4}$$

Thus, we can obtain the optimal $\mathcal{N}_i$ with respect to the MDL principle simply by choosing the π_i which minimizes ((4.3) + (4.4)).

Note that for $\mathcal{N}_i$ $(i = 1, \cdots, n)$, when $\pi_i = \phi$, i.e. $\mathcal{S}_i = \{i\}$, let $N_j(i)$ be the number of examples in which X_i is in $C_i^{(j)}$, and the description length of $\mathcal{N}_i$ for given examples can be written as follows:

$$-\sum_{1\le j\le m_i} N_j(i)\log\hat{p}_j(i),$$

where

$$\hat{p}_j(i) \;=\; \frac{N_j(i)+1}{\sum_{1\le j\le m_i} N_j(i)+m_i}.$$

Definition 4.3 (Description Length of Partial Probabilistic Network with Finite Partitionings) (Mamitsuka, 1995) Let $\mathcal{N}$ be a probablistic network with finite partitionings. For each $i = 1, \cdots, n$, let $\mathcal{N}_i$ be a partial probabilistic network with finite partitionings consisting of partial graph $K_i = (\mathcal{S}_i, \mathcal{A}_i)$, random variables X_t on domains $\mathcal{X}_t$ for each $t \in \mathcal{S}_i$, and finite partitionings $\{C_t^{(j)}\}_{1\le j\le m_t}$ of $\mathcal{X}_t$ for each $t \in \mathcal{S}_i$. Let $N_{j_0,j_1,\cdots,j_{k_i}}(i,\pi_i)$ be the number of examples in which X_i is in $C_i^{(j_0)}$ and $X_{\pi_i^{(t)}}$ is in $C^{(j_t)}_{\pi_i^{(t)}}$ for each $t = 1, \cdots, k_i$. Let $\hat{p}_{j_0|j_1,\cdots,j_{k_i}}(i|\pi_i)$ be the Laplace estimator. We define the description length of $\mathcal{N}_i$ as follows.

$$\sum_{1\le j_0\le m_i}\sum_{1\le j_1\le m_{\pi_i^{(1)}}}\cdots\sum_{1\le j_{k(i)}\le m_{\pi_i^{(k_i)}}} \left(-N_{j_0,j_1,\cdots,j_{k_i}}(i,\pi_i)\log\hat{p}_{j_0|j_1,\cdots,j_{k_i}}(i|\pi_i) + \frac{\log N_{j_0,j_1,\cdots,j_{k_i}}(i,\pi_i)}{2}\right).$$

□

Greedy Algorithm

When we test whether a partial network $\mathcal{N}_i$ is optimal with respect to the MDL principle or not, there exist exponential combinations of the predecessors of a node i in $\mathcal{N}_i$. In addition, there is a no-cycle constraint in the directed acyclic graph of our probabilistic network with finite partitionings. Thus, we have to search for the optimal predecessors of a node, from the exponential combinations of the predecessors, satisfying the no-cycle constraint of the directed acyclic graph. Here, we propose to employ a greedy algorithm which allows us to both obtain a near-optimal network with respect to the MDL criterion and solve the difficulty. The algorithm can be summarized as follows. We start with a network with no arcs. We iteratively add an arc to the network which reduces the description length by the largest amount, among all possible additions which do not violate the no-cycle constraint. When the description length of the network cannot be reduced by adding any arcs, the algorithm terminates.

For a partial network $\mathcal{N}_i$ $(i = 1, \cdots, n)$, let o_i be the node which reduces the description length Γ_i by the largest amount when it is added to π_i among all possible additions. Let $\Delta\Gamma_i$ be the amount to be reduced if o_i is added to π_i. Below, we show our overall algorithm to obtain a near-optimal network efficiently in terms of computation time. In Algorithm 4.4, note that negative examples are used only in step 1.

Algorithm 4.4 (Learning Probabilistic Network with Finite Partitionings) (Mamitsuka, 1995)

```
input: positive and negative examples
output: a trained probabilistic network with finite partitionings
1: For each $i = 1, \cdots, n$, obtain the optimal finite partitionings according to Algorithm 4.2.
2: Calculate initial $\Gamma_i$ for each $i = 1, \cdots, n$, assuming that each node has no predecessors
3: Repeat the following two steps until $\Gamma_i$ is not reduced for $i = 1, \cdots, n$
3-1: For each $i = 1, \cdots, n$, calculate $\Delta\Gamma_i$ and $o_i$.
3-2: Find the maximum of $\Delta\Gamma_i$ $(i = 1, \cdots, n)$ and if $t = arg \max_{1 \le i \le n} \Delta\Gamma_i$, add $o_t$ to $\pi_t$ and $\Gamma_t := \Gamma_t - \Delta\Gamma_t$.
4: Obtain $\pi_i$ $(i = 1, \cdots, n)$ and calculate the conditional probabilities.    □
```

4.3 Experimental Results

We used the *EF-hand* motif. The EF-hand motif is typically found in calcium binding proteins, such as calmodulin, parvalbumin and troponin C, and its function is regulated by one or more calcium atoms (Moncrief et al., 1990; Nakayama et al., 1992). This motif is composed of two α-helices and a loop region connecting them. The loop region, which is called the *EF-hand loop*, binds to a calcium atom and contains twelve amino acids. In Figures 4.1 and 4.2, we show stereo-diagrams of two EF-hand loops in calmodulin, which is noted as 1clm in the PDB database (Bernstein et al., 1977). In Figures 4.1 and 4.2, the loop corresponds to residues 17 to 28 and 126 to 137 in 1clm, respectively, and each thick line shows the side chain of a residue. We represented the EF-hand loop with our probabilistic network with finite partitionings, because the loop was well studied, and hence was an appropriate subject for demonstrating the effectiveness of our method.

4.3.1 Data – EF-hand Motif

From the Swiss-Prot protein sequence database release 28.0 (Bairoch & Apweiler, 1996), we obtained 579 sequences as positive examples of the EF-hand loop, each of which satisfies the following two conditions: 1) a sequence which is noted as the calcium-binding region of an EF-hand motif in the

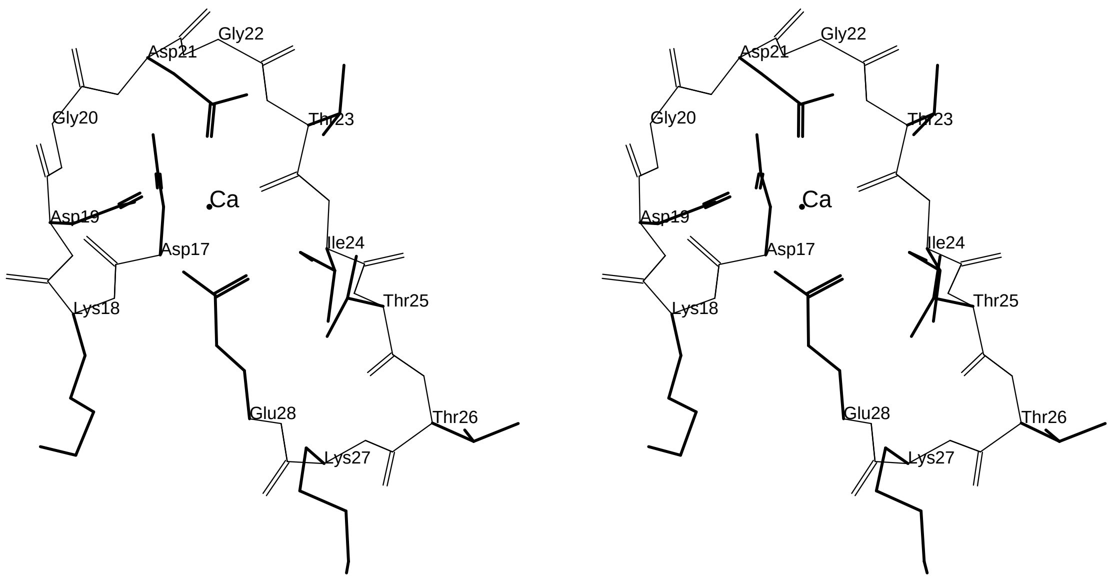


Figure 4.1: First EF-hand loop in 1clm

database, and 2) a sequence which is twelve residues long. The name of the proteins from which the positive examples are derived and the number of positive examples for each name are shown in Table 4.1. Negative examples we used are 579 sequences, each of which is twelve residues long, and were randomly selected from sequences with known structures, so as not to include any positive examples.

4.3.2 Training Probabilistic Network with Finite Partitionings

Finite Partitioning

For $\mathcal{X}_i$ $(i = 1, \cdots, n)$, we used a two-dimensional domain defined by two physico-chemical properties of amino acids, i.e. hydrophobicity and molecular weight. The two-dimensional domain used in our experiments corresponds to the domain $\mathcal{X}_B$ in Chapter 3, which is shown in Figure 3.5. We can reduce the number of amino acids to less than twenty by dividing the domain into several cells.

Probabilistic Network with Finite Partitionings

For simplicity, hereafter we refer to a probabilistic network with finite partitionings as a probabilistic network. The probabilistic network we used has twelve nodes, each of which corresponds to a residue position in the EF-hand loop. More concretely, node i in the probabilistic network corresponds to residue position i in the loop. Henceforth, we write position i for node i. We placed an upper limit of two on the number of predecessors for each position when we train our network. This constraint is reasonable, because the number of examples obtained here is not large enough to calculate joint (and conditional) probabilities for more than three variables, and because any relations between variables, i.e. inter-residue relations for amino acid sequences, will be sufficiently represented even under this constraint.

Figure 4.3 shows the network which was constructed under the constraint, using our learning

Protein	# examples
Calmodulin or Calmodulin related protein	169
Troponin C	62
Parvalbumin α or β	60
Myosin regulatory light chain	30
Calpain	24
Vitamin D dependent calcium-binding protein	24
Sarcoplasmin calcium-binding protein	23
Calcineurin B	12
Spec 1A, 2A or 2C protein	12
Calretinin	11
α-actinin	8
Calcium-dependent protein kinase	8
LPS1 protein	8
S-100 protein	8
Aequorin	6
Flagellar calcium-binding protein	6
Neurocalcin β	6
Visinin	6
Calcium-binding protein	5
Fimbrin (Plastin)	5
15 KD Calcium-binding protein	4
22 KD Calmodulin-like calcium-binding protein	4
23 KD Calcium-binding protein	4
25 KD Calcium-binding protein	4
Caltractin	4
Calcyphosin	4
Diacylglycerol kinase	4
Oncomodulin	4
Optic lobe calcium-binding protein	4
Plasmodial specific protein	4
Probable calcium-binding protein	4
Sorcin	4
Calcyclin	3
Calgizzarin	3
Calgranulin B	3
Luciferin binding protein	3
Neuron specific calcium-binding protein	3
Osteonectin precursor	3
Placental calcium-binding protein	3
20 KD Calcium-binding protein	2
Calgranulin A	2
Calcium vector protein	2
Cell division control protein	2
Recoverin	2
Spectrin α-chain	2
Chemotactic cytokin	1
Matrix glycoprotein sc1 precursor	1
Protein MRP-126	1
QR1 protein precursor	1
S-modulin	1

Table 4.1: 579 EF-hand loop regions obtained from the Swiss-Prot database, Release 28.0

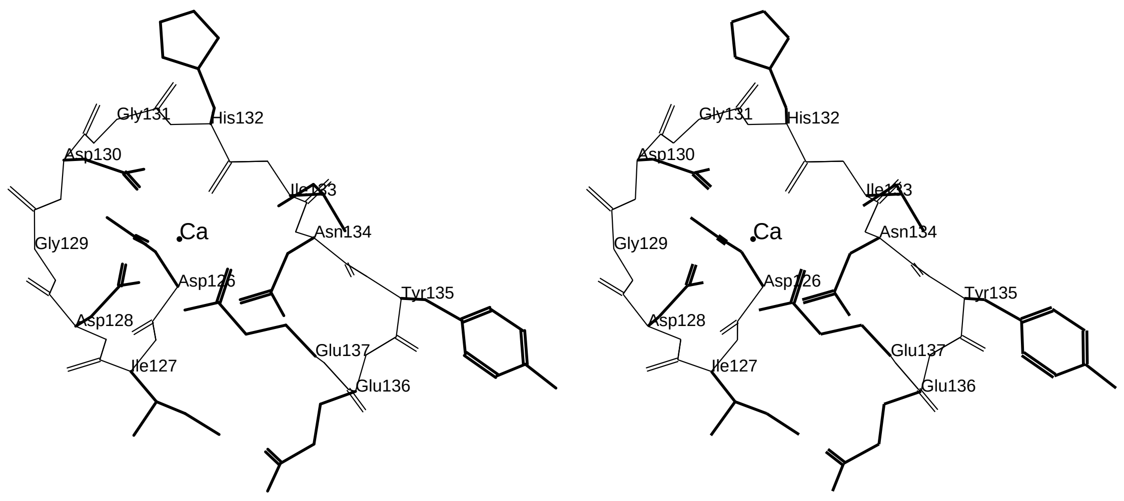


Figure 4.2: Fourth EF-hand loop in 1clm

method. It took less than one second to train the probabilistic network by our method, on a Silicon Graphics Indigo II graphic workstation. Table 4.2 shows (at maximum) the five largest conditional probabilities in each partial probabilistic network with finite partitionings, which are greater than 0.3. Henceforth, we refer to a partial probabilistic network with finite partitionings as a partial network. In Table 4.2, each upper-case character indicates one amino acid letter code. Each partial network in Figure 4.3 is related to one of several structural or functional features in the EF-hand loop. Here we explain two features captured by the network obtained.

First, the partial network of positions 3, 1 and 12 corresponds to the fact that, as shown in Figures 4.1 and 4.2, amino acids of these three positions extend their side chains to a calcium atom, and that the positions are crucial in binding the EF-hand loop to the calcium atom. Interestingly, major conditional probabilities of the partial network shown in Table 4.2, indicate that two separate correlations exist between positions 3 and 12, that is, that amino acid D is at position 3 when amino acid E is at position 12, while amino acid N is at position 3 when amino acid D is at position 12.

Second, the partial network of positions 9, 3 and 5 also reflects the fact that the positions are important as residues binding to the calcium atom. As shown in Table 4.2, the conditional probabilities in the partial network also indicate that there are several separate correlations between position 9 and positions 3 and 5. This type of correlation has not been automatically detected by any other methods based on sequence patterns such as sequence profiles (Lüthy et al., 1994).

The PROSITE database (Hofmann et al., 1999) says that six positions, i.e. 1, 3, 5, 7, 9 and 12, are involved in the calcium-binding. However, the probabilistic network obtained shows that position 7 has no relation to other positions relating to calcium-binding, with the exception of position 9. Actual three-dimensional structures shown in Figures 4.1 and 4.2 indicate that position 7 is binding to the calcium atom not by its side chain but by its backbone carboxyl group. This fact may imply that the probabilistic network trained by our method does not reflect this type of backbone-based binding, probably because the probabilistic network simply focuses on the substitution of amino acid types (i.e. side chains) in relevant positions.

Node(x_i)		Predecessors(π_i)				Conditional probabilities
Position	a.a.	Position	a.a.	Position	a.a.	$\hat{p}(x_i\|\pi_i)$
1	D					0.97
2	K	10	T	9	T	0.67
	K		A		S	0.53
	Q		K		D	0.45
	I		Y		N	0.34
	A		F		D	0.34
3	D	1	D	12	E	0.75
	N		D		D	0.45
4	G	3	D	5	D	0.75
	G		D		N	0.64
	G		D		G	0.42
	G		D		S	0.38
	K or R		D		S	0.34
5	D	3	N			0.80
	D		D			0.38
	S		D			0.30
6	C or G	3	D			0.93
	C or G		N			0.80
7	Q	10	Y	9	N	0.61
	T		F		D	0.60
	E or K		L or V		G	0.57
	F		E		E	0.50
	F		A		S	0.45
8	I	1	D			0.66
9	D	5	D	3	N	0.49
	S		N		D	0.42
	D		N		D	0.35
	E		S		D	0.33
	D		S		D	0.31
	S		G		D	0.31
10	F	9	D	4	G	0.67
	Y		N		G	0.61
	L or V		G		G	0.57
	T		T		G	0.56
	E		E		K or R	0.53
11	A	10	A	5	N	0.67
	K		T		D	0.61
	E		Y		D	0.57
	P		F		N	0.56
	D		E		S	0.35
12	E					0.87

a.a. = Amino acids

Table 4.2: Conditional probabilities of partial probabilistic networks

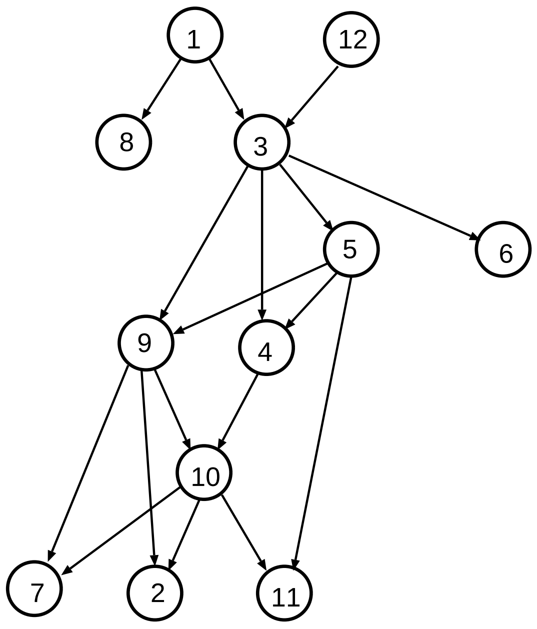


Figure 4.3: Probabilistic network for EF-hand loop

Likelihood Calculation

We can calculate the joint probability for an example given by the network in Figure 4.3, using Eq. (4.1). This probability can be regarded as the likelihood that a given example is an EF-hand loop.

A histogram of the logarithmic likelihoods calculated for positive and negative examples used in training our method is shown in Figure 4.4. As shown in the figure, the probabilistic network clearly separates positive examples, the logarithmic likelihoods of which are distributed over a range of -2 to -12, from negative examples which correspond to the right part of the histogram.

4.3.3 Comparing Probabilistic Network with Neural Network in Cross-validation Test

With a type of cross-validation test, we evaluated our method by comparing it with feed-forward type neural networks. In the cross-validation test, we first divide positive examples into five datasets, and the same procedure is done for negative examples. We next train a probabilistic network with four of the five datasets of positive and negative examples, and in testing, calculate the likelihood for each example in the other remaining dataset. We further repeat this procedure for all five possible cases. The trained network is used as a classifier as follows. When the likelihood of an example is over some given threshold, the example is predicted to be a positive example, and otherwise, the example is predicted to be a negative one. The threshold is set so that the number of errors is minimized.

Here we briefly explain the neural network which was used for comparing with our probabilistic network. The network has three layers which consist of twelve input windows, 40 hidden units and two output units, which are called a positive unit and a negative unit. Each of the twelve input windows, which corresponds to a residue in each example, has twenty input units. When an amino acid is given to an input window, the input unit corresponding to that acid outputs 1 and all other input units in the window output 0. When, in training, a positive example is given, 1 and 0 are given to the positive

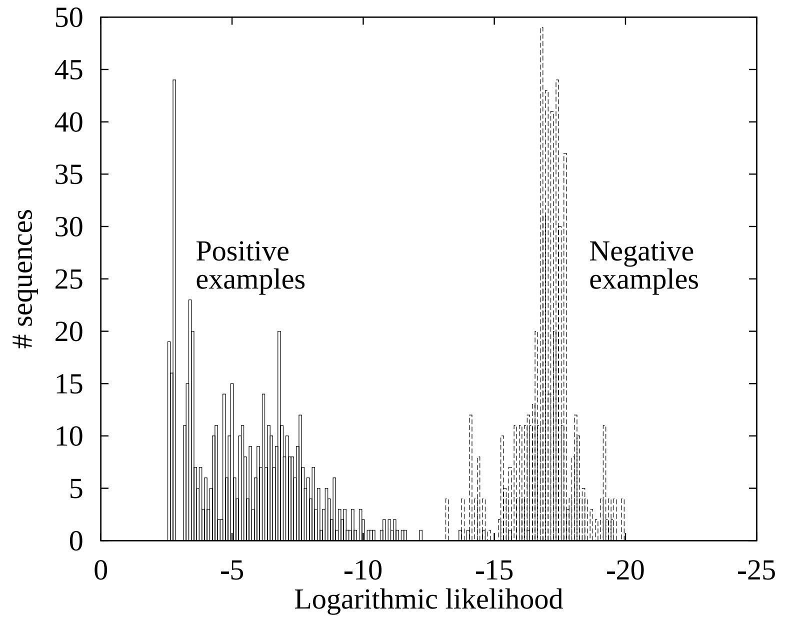


Figure 4.4: Discrimination histogram for examples

	Probabilistic networks	Neural networks
1	99.7%	99.7%
2	99.1%	99.7%
3	100%	100%
4	98.2%	98.8%
5	99.7%	100%
Average accuracy	99.1%	99.5%

Table 4.3: Prediction accuracies in cross-validation

and negative unit, respectively, as teaching signals, and otherwise 0 and 1 are given. In prediction, a given sequence is predicted to be a positive example if the value outputted by the positive unit is larger than that by the negative unit, and otherwise the sequence is predicted to be a negative one. We used a standard back-propagation learning method (e.g. Rumelhart et al., 1986) for the neural network. The neural network used here is similar to the ones used by Bohr et al. (1988) and Hayward and Collins (1992) to predict whether a residue is in an α-helix or not.

Table 4.3 shows the result of the cross-validation test for our method and the neural network learning method. The table shows that the average prediction accuracy of our probablistic networks reached more than 99%, which is almost the same as that of neural networks. We emphasize that our probabilistic network can provide a visual aid to understanding inter-residue relations in given examples, which are quantified with conditional probabilities, and that the network can be obtained within only one second. The representation and speed of our method are at a level which the neural-network learning method cannot match.

4.4 Conclusion

We defined a new probabilistic network, called a probabilistic network with finite partitionings, for representing inter-residue relations within a biological sequence, and established a new efficient learning algorithm of the probabilistic network. In our experiment, we used actual sequences of the EF-hand loop motif to learn our probabilistic network. Experimental results show that our method allowed us to learn a probabilistic network of the motif within only one second, and that the network obtained gave a number of inter-residue relations, which are quantified with conditional probabilities, and each of which corresponds to a crucial feature of the motif. In particular, the network shows that positions related to binding to a calcium atom, namely positions 1, 3, 5, 9, and 12, have mutual inter-residue relations, and conditional probabilities attached to the relations imply that several separate correlations are hidden in each of the relations. These results indicate that our method is useful for finding important inter-residue relations in motif sequences.

Chapter 5

Supervised Learning of Hidden Markov Models

5.1 Introduction

Hidden Markov Models (hereafter referred to as HMMs) have been proven to be a useful tool for a number of subjects in computational molecular biology (Eddy, 1996; Durbin et al., 1998), such as protein modeling (Stultz, White, & Smith, 1993; White, Smith, & Smith, 1994; Krogh et al., 1994a; Baldi et al., 1994; Baldi & Chauvin, 1994a), gene recognition (Churchill, 1989; Krogh, Mian, & Haussler, 1994b; Krogh, 1997), and secondary structure prediction (Asai, Hayamizu, & Handa, 1993). Among these various applications, the most popular application of an HMM in this field has been the construction of an HMM-based profile to represent multiple biological sequences belonging to a single class such as a superfamily. The main theme in generating such a profile is to improve the sensitivity of the model to achieve high accuracy for new sequences in several tasks such as database search or data validation. In the molecular biology field, however, this theme is often subject to a strong constraint, in which the number of available biological sequences belonging to a single class is extremely small, since the sequences can be obtained only through money- and time-consuming biochemical experiments.

As mentioned in Chapter 2, in training parameters of an HMM, there is a widely used and efficient learning method, called the Baum-Welch or Forward-Backward (Baum, 1972). The Baum-Welch algorithm is based on EM (Expectation-Maximization) algorithm (Dempster, Laird, & Rubin, 1977) which trains a given model so that the likelihoods of the training sequences are maximized. However, for the situation in which only a small training dataset is given, it had been pointed out that the Baum-Welch algorithm cannot always provide sufficient discrimination ability (Brown et al., 1993). In contrast to the Baum-Welch's reestimation rules which let the parameters skip about within their space, Baldi et al. (Baldi & Chauvin, 1994b) proposed a smooth learning algorithm which gradiently optimizes parameters and can be executed by even on-line learning. This smooth algorithm might enhance the sensitively of the HMMs trained.

Under these circumstances, we established a new learning method for training an HMM of a certain biological class not only from examples belonging to the class but also from examples which do not (Mamitsuka, 1996, 1997). In our method, we first change the probability parameters of a given HMM into real-valued parameters and prepare an *error-distance function* which measures an error-distance between the real likelihood and the target likelihood for a given training sequence. To train the parameters of an HMM, we then use a gradient descent learning algorithm (Fletcher, 1987) to minimize the energy function prepared. In other words, our method implement new *supervised learning* of an HMM while the Baum-Welch is based exclusively on unsupervised learning. Furthermore, our method is a smooth algorithm in parameter optimization, with a computation time per iteration in learning on the same order as that of the Baum-Welch algorithm.

Before we established our new method, several methods had been proposed in the computational molecular biology field to improve the sensitivity of an HMM for unknown sequence data. Krogh and Mitchison (1995) and Eddy et al. (1995) independently proposed original learning methods for stochastic models to solve the problem that the probability distribution trained by usual maximum likelihood criterion is likely to pick out minor data included in training examples. They used only positive examples, i.e. the training examples belonging to the class which should be represented by a model, and using the criterion called maximum discrimination or maximum entropy, they introduced slight modification of Baum-Welch's re-estimation rules in training an HMM to correct amino acid biases included in given data. Note that in proposing our method, we wished to consider a new learning method itself which is both applicable to any HMM structure and also improves the discrimination accuracy for unknown sequences so that it out-performs conventional methods such as Baum-Welch.

We summarize the original characteristics of our method as follows:

1) *Supervised learning of hidden Markov models.*
   Our learning method allows us to train HMMs with a supervised learning algorithm, namely, to train HMMs using data consisting of a number of discrete or continuous classes or data having real-valued labels. However, in the context of natural language processing which is the major application field of HMMs, most of the learning algorithms proposed to train HMMs are based on unsupervised learning, and Bahl et al. (1986), which proposed a supervised learning algorithm for HMMs, has been a unique work. Their method trains HMMs with both positive and negative examples of a given class, based on the maximum mutual information criterion, but it cannot deal with data consisting of continuous classes or data having real-valued labels. Thus, no supervised learning algorithm able to deal with such data, had ever been proposed in training HMMs until we established our method, to the best of our knowledge.

2) *Fully-connected models tested.*
   In the field of natural language processing and computational molecular biology, in each of which HMMs have been extensively used, particularly wide usage has been made of a type of HMM, called a left-to-right HMM, in which any arc has to go from a state A to a state B in the right side of A. On the other hand, in our experiments, we used a *fully-connected HMM* in which an arc which starts from a state may go to any state. That is, we used a flexible model which is free of any constraint. In other words, we can say that in our experiment, we attempted to learn not only the probability parameters of an HMM, but also the structure of the HMM itself.

We performed two experiments. In the first experiment, we used the lipocalin family motif, and generated both positive examples, each of which corresponds to a part of a lipocalin sequence, and negative ones, each of which does not belong to the lipocalin family but has the motif. We evaluated our method with a type of cross-validation test using fully-connected HMMs with six to fourteen states, and compared the results obtained with those of the Baum-Welch and Baldi algorithms. The results indicate that our method makes fewer discrimination errors than the other two methods. From these results, we conclude that the use of negative examples is effective in learning the parameters of an HMM, and that our method is useful for training an HMM with both positive and negative examples.

The second experiment actually consisted of two experiments. In the first, we verified the discrimination ability of our supervised learning method compared with that of two other methods, i.e. a back-propagation neural network and the Baum-Welch learning of an HMM. In this experiment, we used actual peptide data in association with their real-valued ability to activate T-cell proliferation, which data was obtained from the MHCPEP database developed by Brusic et al. (1997). The experiment was performed by conducting a cross-validation test while varying the proportion of training data to all obtained data. The result of this experiment shows that at any proportion of training data to all data, the average discrimination accuracy of our method was approximately $2 - 15\%$ better than other methods, i.e. the Baum-Welch reestimation for fully-connected or alignment HMMs and the back-propagation neural network, which had been regarded as the most accurate method in predicting MHC binding peptides. From this result, we conclude that our method was superior in discrimination ability to conventional methods in the field of predicting peptides that bind to MHC.

Second, from the MHCPEP database, we obtained peptides which bind to an MHC protein, called HLA-A2, and trained 100 HMMs using the peptides with our supervised learning algorithm. Out of the 100 models trained for HLA-A2, we chose the one which could explain the data best and showed the HMM. Using the model trained by the data of HLA-A2, we randomly generated peptides which are expected to have a high ability to bind to HLA-A2, but which are not yet known. From this experiment, we ascertained that an HMM trained by our algorithm captures sequence patterns in training data separately, and found that the extracted patterns include not only existing motifs of MHC binding peptides but also new sequence patterns, each of which characterizes a part of the training peptides.

A brief note on the latest information regarding supervised learning of HMMs: No new method on supervised learning of an HMM has been proposed since we established our method, but Krogh and Riis proposed a new method for supervised learning of an HMM, called a class HMM (CHMM) (Krogh & Riis, 1999). The CHMM uses a number of techniques which are the same as those used in our method and it can train an HMM using symbol sequences which are categorized in one or more classes (labels); however the CHMM has a disadvantage in that it cannot deal with continuous classes. In contrast, our learning algorithm train an HMM using sequences with not only continuous classes but also real-valued labels. From this viewpoint, we can say that even now, our method has a significant advantage in the field of training algorithms of HMMs.

5.2 Hidden Markov Models

We recall the definition of hidden Markov model (HMM), forward and backward probabilities, and ξ and γ, which are given in Chapter 2.

Definition 5.1 (Hidden Markov Model) A *hidden Markov model* H is a 7-tuple $\langle \Sigma, Q, I, F, E, A, B \rangle$, where:
(i) Σ is a finite alphabet.
(ii) Q is a finite set of *states*.
(iii) $I \subseteq Q$ is a finite set of *initial states*.
(iv) $F \subseteq Q$ is a finite set of *final states*.
(v) $E \subseteq Q \times Q$ is a finite set of *arcs*. An arc $(i, j) \in E$ is also denoted by e_{ij}.
(vi) $A = (a_{ij})_{i,j \in Q}$ is a matrix of *state transition probabilities*, i.e. a_{ij} is the probability that is attached to arc e_{ij} which indicates the transition from state i to j, where $\sum_{j \in Q} a_{ij} = 1$ is assumed for each $i \in Q$.
(vii) $B = (b_j(c))_{j \in Q, c \in \Sigma}$ is a matrix of *symbol output probabilities*, i.e. $b_j(c)$ is the probability that is attached to state j which indicates that state j outputs symbol c, where $\sum_{c \in \Sigma} b_j(c) = 1$ is assumed for each $j \in Q$. □

Definition 5.2 (Forward Probability) (Rabiner, 1989) Let $H = \langle \Sigma, Q, I, F, E, A, B \rangle$ be an HMM. For convenience, let $Q = \{1, \cdots, M\}$, $\Sigma = \{1, \cdots, L\}$. For a symbol sequence $\sigma = \sigma_1 \cdots \sigma_{n_\sigma}$, $1 \le t \le n_\sigma$ and a state j, the *forward probability* $\alpha_\sigma[t, j]$ is the probability that the partial sequence $\sigma_1 \cdots \sigma_t$ is generated, and that the state at time t is j. For $t = 0$ and a state j, the forward probability $\alpha_\sigma[0, j]$ is the probability that no symbol is generated, and that the state at time 0 is j. □

Definition 5.3 (Backward Probability) (Rabiner, 1989) Let $H = \langle \Sigma, Q, I, F, E, A, B \rangle$ be an HMM. For convenience, let $Q = \{1, \cdots, M\}$, $\Sigma = \{1, \cdots, L\}$. For a symbol sequence $\sigma = \sigma_1 \cdots \sigma_{n_\sigma}$, $0 \le t \le n_\sigma - 1$ and a state i, the *backward probability* $\beta_\sigma[t, i]$ is the probability that the partial sequence $\sigma_{t+1} \cdots \sigma_{n_\sigma}$ is generated, and that the state at time t is i. For $t = n_\sigma$ and a state i, the backward probability $\beta_\sigma[n_\sigma, i]$ is the probability that no symbol is generated, and that the state at time n_σ is i. □

Note that $\sum_{1 \le i \le M} \alpha_\sigma[n_\sigma + 1, i]\beta_\sigma[n_\sigma + 1, i]$, as well as $\sum_{1 \le i \le M} \alpha_\sigma[0, i]\beta_\sigma[0, i]$, corresponds to the likelihood that the given training sequence σ is generated by the HMM H, i.e. $P(\sigma|H)$.

Our learning algorithm, as well as the Baum-Welch algorithm, uses the following two probabilities ξ and γ.

Definition 5.4 (ξ and γ) (Rabiner, 1989) Let $H = \langle \Sigma, Q, I, F, E, A, B \rangle$ be an HMM. For convenience, let $Q = \{1, \cdots, M\}$, $\Sigma = \{1, \cdots, L\}$. For a symbol sequence $\sigma = \sigma_1 \cdots \sigma_{n_\sigma}$, $0 \le t \le n_\sigma - 1$ and two states i and j, $\xi_\sigma[t, i, j]$ is the probability that the σ is generated by the H, and that the two states at times t and $t + 1$ are i and j, respectively. Similarly, for a symbol sequence $\sigma = \sigma_1 \cdots \sigma_{n_\sigma}$, $0 \le t \le n_\sigma$ and a state i, $\gamma_\sigma[t, i]$ is the probability that the σ is generated by the H, and that the state at time t is i. □

The probabilities ξ and γ are calculated using the forward and backward probabilities as follows:

$$\xi_\sigma[t, i, j] = \frac{\alpha_\sigma[t, i] a_{ij} b_j(\sigma_{t+1}) \beta_\sigma[t + 1, j]}{P(\sigma|H)} \quad (0 \le t \le n_\sigma - 1), \tag{5.1}$$

$$\gamma_\sigma[t, i] = \frac{\alpha_\sigma[t, i]\beta_\sigma[t, i]}{P(\sigma|H)} \quad (0 \le t \le n_\sigma). \tag{5.2}$$

Proposition 5.5 (Time Complexity of Forward, Backward, ξ and γ)
The time complexity of the forward and backward algorithms for an HMM is $O(N \cdot M^2)$, where N is the length of a given sequence and M is the number of states in the HMM. The time complexity of calculating ξ and γ is also $O(N \cdot M^2)$.
(Proof)
The time complexity of the forward and backward algorithm depends on the calculation of step 2 in Algorithm 2.5 and step 3 in Algorithm 2.15. The time complexity of calculating γ and ξ also depends on them. □

5.3 Supervised Learning Algorithm for Hidden Markov Models

We introduce real-valued parameters w_{ij} and $v_j(c)$, which are used in (Bridle, 1990) and (Baldi & Chauvin, 1994b), as follows:

$$a_{ij} = \frac{e^{\lambda_1 w_{ij}}}{\sum_k e^{\lambda_1 w_{ik}}}, \quad b_j(c) = \frac{e^{\lambda_2 v_j(c)}}{\sum_k e^{\lambda_2 v_j(k)}}, \tag{5.3}$$

$$(\sum_{1 \le j \le M} a_{ij} = 1,\ a_{ij} \ge 0, \quad \sum_{1 \le c \le L} b_j(c) = 1,\ b_j(c) \ge 0.)$$

where λ_1 and λ_2 are constants. The equations allow w_{ij} and $v_j(c)$ to be any real-value, satisfying the constraints of a_{ij} and $b_j(c)$ given by Definition 5.1.

We define an error-distance function.

Definition 5.6 (Error-distance Function) (Mamitsuka, 1996) Let $H = \langle \Sigma, Q, I, F, E, A, B \rangle$ be an HMM. Let Ξ be a set of symbol sequences. Let $p_\sigma \in [0,1]$ be the likelihood that a sequence $\sigma \in \Xi$ is generated by an H, i.e. $P(\sigma|H)$, and we call p_σ the *real likelihood*. Let $p^*_\sigma \in [0,1]$ be a real-value which we call the *target likelihood*. For a sequence σ, let D_σ be a positive real-value given by the following form:

$$D_\sigma = (\log(\frac{p^*_\sigma}{p_\sigma}))^2.$$

For a sequence σ, let $g_\sigma \in [0,1]$ be a real-value given as follows:

$$g_\sigma \quad = \quad \frac{D_{max} - D_\sigma}{D_{max}},$$

where D_{max} is a fixed positive real-value, which is larger than $D_{\sigma|\sigma\in\Xi}$.

For a set of sequences Ξ, we define an *error-distance function* $F(\Xi) \in [0,\infty]$ as follows:

$$F(\Xi) \quad = \quad \sum_{\sigma\in\Xi} - \log g_\sigma. \tag{5.4}$$

□

For each sequence σ, the purpose of our learning algorithm is that $D_\sigma \to 0$. Here, for each σ, $g_\sigma \to 1$ if $D_\sigma \to 0$. Furthermore, $g_\sigma \to 1$ for each σ means that $F(\Xi) \to 0$. Thus, in order to attain the purpose of our learning algorithm, we have to minimize the $F(\Xi)$. In minimizing the $F(\Xi)$, we obtain a smooth learning algorithm for optimizing real-valued parameters, w_{ij} and $v_j(c)$, as follows:

Algorithm 5.7 (Supervised Learning for Hidden Markov Models) (Mamitsuka, 1996)
input: a set of symbol sequences Ξ and initial a_{ij} and $b_j(c)$
output: trained a_{ij} and $b_j(c)$
1: Calculate w_{ij} and $v_j(c)$ from the initial a_{ij} and $b_j(c)$, using (5.4).
2: Repeat the following two steps until a stopping condition is satisfied, usually until the changes in the a_{ij} and $b_j(c)$ become smaller than a certain preset amount.
2-1: Update the w_{ij} and $v_j(c)$ using (5.5) and (5.6).
2-2: Calculate a_{ij} and $b_j(c)$ from w_{ij} and $v_j(c)$, using (5.4).

$$\begin{aligned} w_{ij}^{new} &= w_{ij}^{old} + C_a \frac{\partial F(\Xi)}{\partial w_{ij}} \\ &= w_{ij}^{old} + C_a \sum_{\sigma\in\Xi} \frac{d_\sigma}{(D_{max} - D_\sigma)} \sum_{0\le t\le n_\sigma - 1} (\xi_\sigma[t,i,j] - a_{ij}\gamma_\sigma[t,i]), \qquad (5.5) \\ v_j(c)^{new} &= v_j(c)^{old} + C_b \frac{\partial F(\Xi)}{\partial v_j(c)} \\ &= v_j(c)^{old} + C_b \sum_{\sigma\in\Xi} \frac{d_\sigma}{(D_{max} - D_\sigma)} \sum_{1\le t\le n_\sigma} (\gamma_\sigma[t|\sigma_t = c, j] - b_j(c)\gamma_\sigma[t,j]), \qquad (5.6) \end{aligned}$$

where $d_\sigma = \log(\frac{p^*_\sigma}{p_\sigma})$ and C_a and C_b are positive constants. □

We here give two propositions relevant to the partial derivatives of a_{ij}.

Proposition 5.8

$$\frac{\partial p_\sigma}{\partial a_{ik}} = \sum_{0\le t\le n_\sigma-1} \alpha_\sigma[t,i] b_k(\sigma_{t+1}) \beta_\sigma[t+1,k]$$

□

Proposition 5.9

$$\frac{\partial a_{ij}}{\partial w_{ik}} = \lambda a_{ij}(\delta_{jk} - a_{ik}) \ , \qquad \delta_{jk} = \begin{cases} 1 & \text{if } j = k, \\ 0 & \text{otherwise.} \end{cases}$$

□

Then, using the two propositions, we show the derivation of $\frac{\partial F(\Xi)}{\partial w_{ij}}$ in (5.5).

$$\begin{aligned} \frac{\partial F(\Xi)}{\partial w_{ij}} &= \sum_k \sum_s \frac{\partial \log g_\sigma}{\partial a_{ik}} \frac{\partial a_{ik}}{\partial w_{ij}} \\ &= \sum_{\sigma\in\Xi} \frac{2d_\sigma}{p_\sigma(d_{max}^2 - d_\sigma^2)} \sum_k \frac{\partial p_\sigma}{\partial a_{ik}} \frac{\partial a_{ik}}{\partial w_{ij}} \\ &= \lambda \sum_{\sigma\in\Xi} \frac{2d_\sigma}{p_\sigma(d_{max}^2 - d_\sigma^2)} \sum_k (\delta_{kj} - a_{ij}) \sum_{0\le t\le n_\sigma-1} \alpha_\sigma[t,i] a_{ik} b_k(\sigma_{t+1}) \beta_\sigma[t+1,k] \\ &\quad \text{since} \quad \text{Propositions 5.8 and 5.9} \\ &= \lambda \sum_{\sigma\in\Xi} \frac{2d_\sigma}{(d_{max}^2 - d_\sigma^2)} \sum_k (\delta_{kj} - a_{ij}) \sum_{0\le t\le n_\sigma-1} \xi_\sigma[t,i,k] \\ &= Const_a \sum_{\sigma\in\Xi} \frac{d_\sigma}{(d_{max}^2 - d_\sigma^2)} \sum_{0\le t\le n_\sigma-1} (\xi_\sigma[t,i,j] - a_{ij}\gamma_\sigma[t,i]), \end{aligned}$$

where $Const_a$ is a constant.

We can derive (5.6) in a similar manner. In (5.5) and (5.6), if a part, $\frac{d_\sigma}{(D_{max}-D_\sigma)}$, is removed, both equations are equal to those of Baldi's smooth algorithm. Furthermore, from (5.5) and (5.6), we can derive an on-line supervised learning algorithm as done in Baldi and Chauvin (1994b).

Algorithm 5.10 (On-line Supervised Learning for Hidden Markov Models) (Mamitsuka, 1996)
input: a set of symbol sequences Ξ and initial a_{ij} and $b_j(c)$
output: trained a_{ij} and $b_j(c)$
1: Calculate w_{ij} and $v_j(c)$ from the initial a_{ij} and $b_j(c)$, using (5.4).
2: For each sequence σ, repeat the following two steps until a stopping condition is satisfied.
2-1: Update the w_{ij} and $v_j(c)$ using (5.7) and (5.8).
2-2: Calculate a_{ij} and $b_j(c)$ from w_{ij} and $v_j(c)$, using (5.4).

$$w_{ij}^{new} = w_{ij}^{old} + C'_a \frac{d_\sigma}{(D_{max} - D_\sigma)} \sum_{0\le t\le n_\sigma-1} (\xi_\sigma[t,i,j] - a_{ij}\gamma_\sigma[t,i]), \tag{5.7}$$

$$v_j(c)^{new} = v_j(c)^{old} + C'_b \frac{d_\sigma}{(D_{max} - D_\sigma)} \sum_{1\le t\le n_\sigma} (\gamma_\sigma[t|\sigma_t = c, j] - b_j(c)\gamma_\sigma[t,j]), \tag{5.8}$$

where $d_\sigma = \log(\frac{p^*_\sigma}{p_\sigma})$ and C'_a and C'_b are positive constants. □

Proposition 5.11 (Time Complexity of Our Learning Algorithm)
The time complexity of one iteration of our learning algorithm given by Algorithm 5.10 is $O(N \cdot M^2)$, while that given by Algorithm 5.7 is $O(N \cdot M^2 \cdot W)$, where W is the number of symbol sequences, N is the length of a given sequence and M is the number of states in an HMM.
(Proof)
The time complexity of Algorithm 5.10 depends on calculating γ and ξ for a given sequence, and the time complexity of them is given by $O(N \cdot M^2)$ in Proposition 5.5. On the other hand, in Algorithm 5.7, calculating the γ and ξ has to be done for all symbol sequences at once, and then the complexity of the algorithm is given by $O(N \cdot M^2 \cdot W)$. $\square$

5.4 Experimental Results – 1

In this experiment, we evaluated our method (hereafter referred to as MA) by applying it to a sequence classification problem, and compared the results with those of the Baum-Welch (hereafter referred to as BW) and the Baldi algorithm (hereafter referred to as BA). We used here a fully-connected HMM, which has only one initial state and one final state, neither of which output any symbols. That is, the initial state has arcs going to any state except for the final state, and any state except for the final state has arcs going to any state other than the initial state.

Here, let us describe how we set p^*_σ in our experiments. Using Algorithm 2.5, we first estimate the likelihood that a symbol sequence of length n is generated by the fully-connected HMM with uniform distributions to be approximately $\frac{1}{M}(\frac{1}{L})^n$, which is obtained by assuming $a_{ij} = \frac{1}{M}$ and $b_j(c) = \frac{1}{L}$, where M denotes the number of states, and L denotes the number of symbols. Then, we let p^*_σ be $(\frac{1}{20})^{0.01 \times n}$ and $(\frac{1}{20})^{1.99 \times n}$ for positive and negative examples, respectively, to satisfy the above condition.

5.4.1 Data – Lipocalin Family Motif

We used the lipocalin family motif (Peitsch & Boguski, 1991) noted in Release 12.0 of the PROSITE database (Bairoch, P.Bucher, & Hofmann, 1996). The motif is '[DENG] - X - [DENQGSTARK] - X(0,2) - [DENQARK] - [LIVFY] - {CP} - G - {C} - W - [FYWLRH] - X - [LIVMTA]', where no amino acids from among those enclosed in { } are not permitted, but either zero, one or two amino acids are permitted at the X(0,2). Thus, any sequence having the motif is twelve to fourteen residues long. Figure 5.1 shows a three-dimensional structure (Peitsch & Boguski, 1990) of the sequence having the motif in human apolipoprotein which belongs to the lipocalin family, and the structure is constructed from the coordinates noted in the PDB database (Bernstein et al., 1977) as '2apd'. The structure shown in the figure corresponds to a part of a β-strand, and the amino acid sequence is DVNKYLGRWYEI from right to left in the figure. From the Swiss-Prot database Release 29.0 (Bairoch & Apweiler, 1996), we obtained 142 sequences, each of which contains the motif. Of the sequences, we used 99 sequences as positive examples which were included in actual lipocalin family protein sequences, and 43 sequences as negative examples which had the motif but were not in any lipocalin family proteins.

The goal of the problem which we now deal with is to distinguish positive examples from negative ones. Note that the problem is rather difficult in that we have to separate positive examples from false positive examples, since the negative examples as well as positive ones have the motif and we cannot recognize positive examples from all the sequences using the motif only.

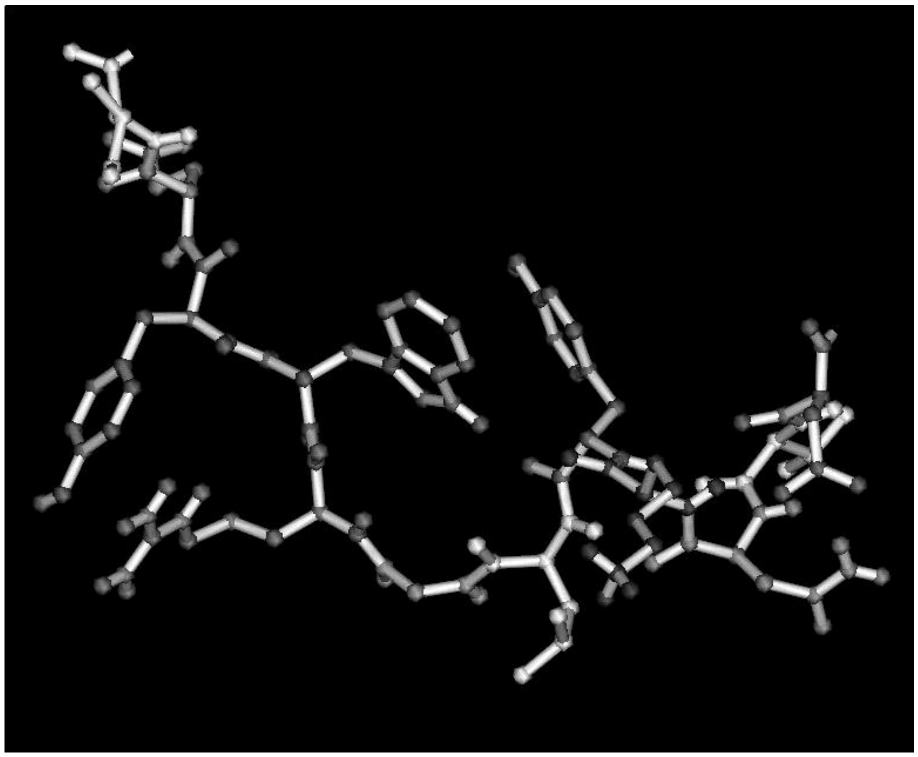


Figure 5.1: 3-dimensional structure of lipocalin motif

5.4.2 Comparing Supervised Learning Method with Other Methods in Cross-validation Test

Though the number of examples obtained was rather small, we conducted a type of cross-validation whose procedure is described as follows: First, both positive and negative examples are randomly divided into two classes, i.e. training and test. In the training phase, we train an HMM with the set of training examples. Using the trained HMM, in the test phase, we try to distinguish positive test examples from negative ones, based on the likelihood of each test sequence. In the test phase, we calculate the modified logarithmic likelihood (hereafter referred to as the MLL) of each sequence to compare test sequences having a variety of lengths with each other, and the MLL of each sequence is calculated by dividing the log-likelihood of the sequence by its length. We randomly generate a pair of training and test examples five times. For each set of training examples, we repeat a pair of training and test phases five times with a variety of different random initial values. The performance of a given learning method is evaluated by the average performance over the 25 trials.

The HMMs used in the cross-validation test had between six and fourteen states. When we tried to separate positive examples from negative ones using a trained HMM in the test phase of the cross-validation, the discrimination ability of the HMM was measured by the minimum number of errors (hereafter referred to as MNE) which can be obtained while changing a cut-off value for the MLLs of test sequences, which classifies test examples into two classes, i.e. positive and negative. Figure 5.2 shows the average MNE over all the repeated trials in the cross-validation for the three methods, i.e. BW, BA and MA. As shown in the figure, in any HMM size, MA made fewer discrimination errors than the other two methods. In particular, in MA, the average MNE of HMMs with fourteen

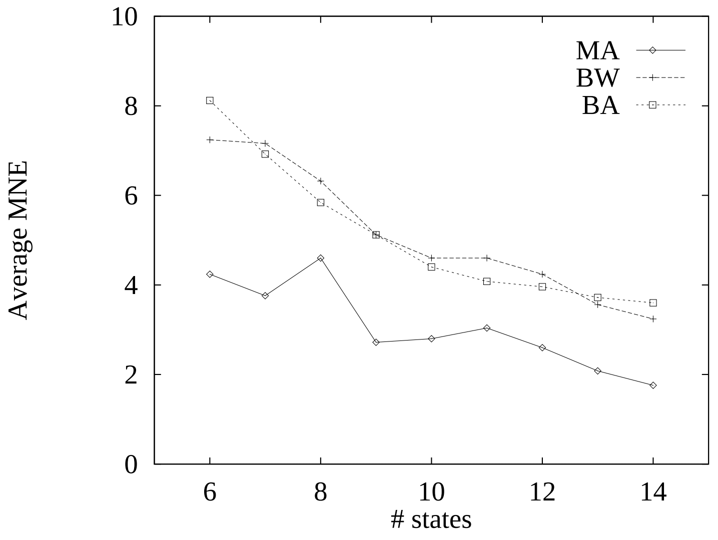


Figure 5.2: Average MNE in cross-validation test for lipocalin motif

states dropped to less than two, which was the minimum of all cases of the three methods. On the other hand, there was not much to choose between BA and BW, but BA appears to be slightly better than BW. From this result, we can easily guess that the use of negative examples in MA reduces the average MNE. Figure 5.3 shows the distributions of MNE of HMMs with no less than ten states, for the three methods in the same experiment. This result also indicates that MA surpasses the other two methods. Specifically, the MNE of MA converged at two while that of the other methods was fairly widely spread over a range of from two or three to six.

Furthermore, using this cross-validation framework, we conducted two types of experiments for BW and BA, and also compared the results with those of MA in terms of the average MNE. In the first experiment (hereafter referred to as 'Exp1'), we prepared two HMMs having a common structure, which were trained by using positive and negative examples, individually. For a given test example, we can calculate the difference between the two MLLs obtained from the two HMMs. Using the difference for each test example, we estimated the MNE for a given set of test examples. In the second experiment (hereafter referred to as 'Exp2'), as initial parameters of HMMs, we used the probability parameters of the HMMs which had already been trained by MA and trained them with the other methods using positive examples only. The MLL for an input test example is provided with the trained HMM, and we then used the MLLs to calculate the MNE for a given set of examples. The results of these two experiments were evaluated by the average MNE.

Figure 5.4 shows the average MNE of Exp1 and Exp2 in which both BW and BA were used as learning methods. As shown in the figure, for both BW and BA, the average MNE of Exp1 was somewhat improved over the results shown in Figure 5.2. However, the values were still greatly inferior to those for MA, even though two HMMs (trained by positive and negative examples respectively) were used. These results indicate that this way of handling negative examples is rather less effective than MA. The Exp2 results were also worse than those obtained with MA, but interestingly, far surpassed both the BW and BA results. Note here that these two methods used positive examples only while MA used both positive and negative ones, and hence it is likely that the constraints in the parameter space of these two are rather different from those of MA. In other words, learning with

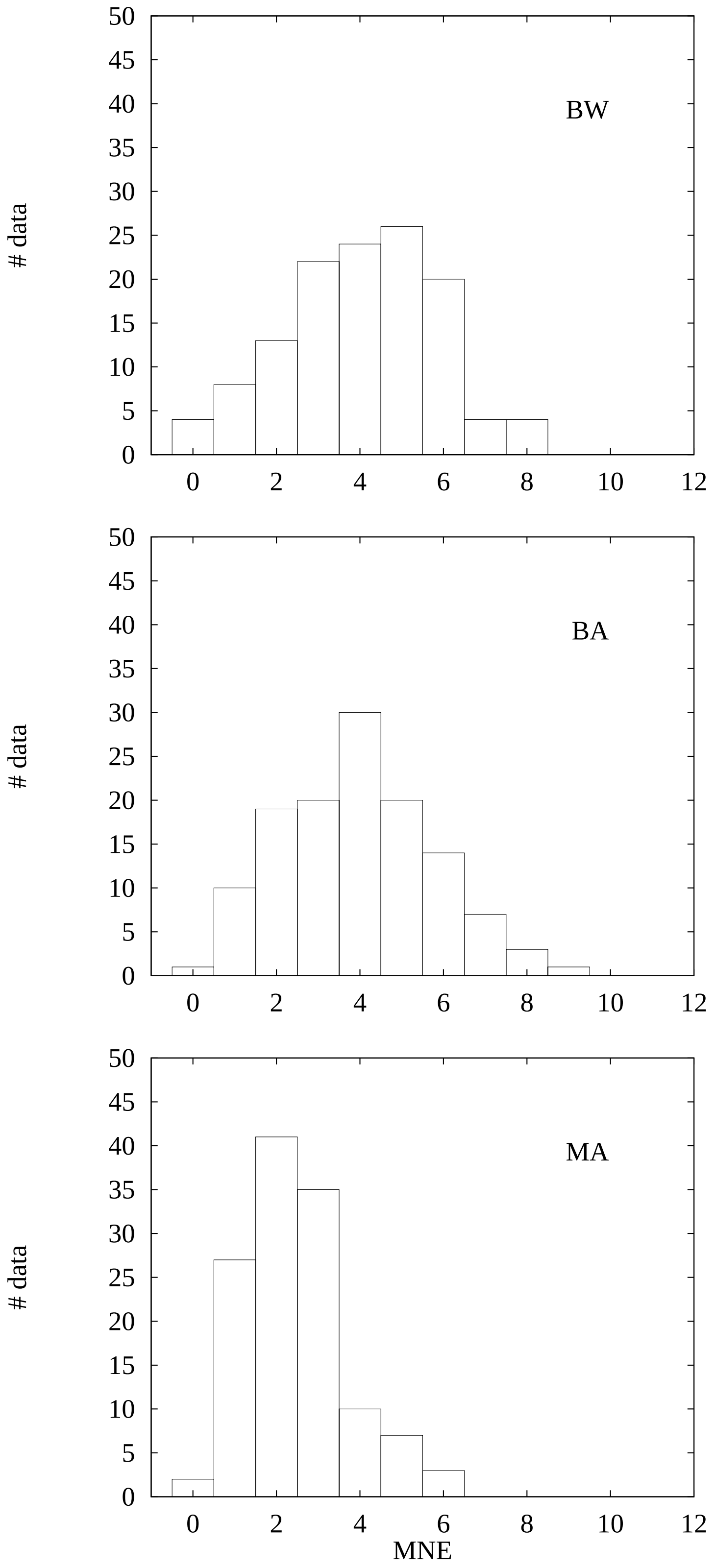


Figure 5.3: Error distributions of three learning methods in cross-validation test for lipocalin motif

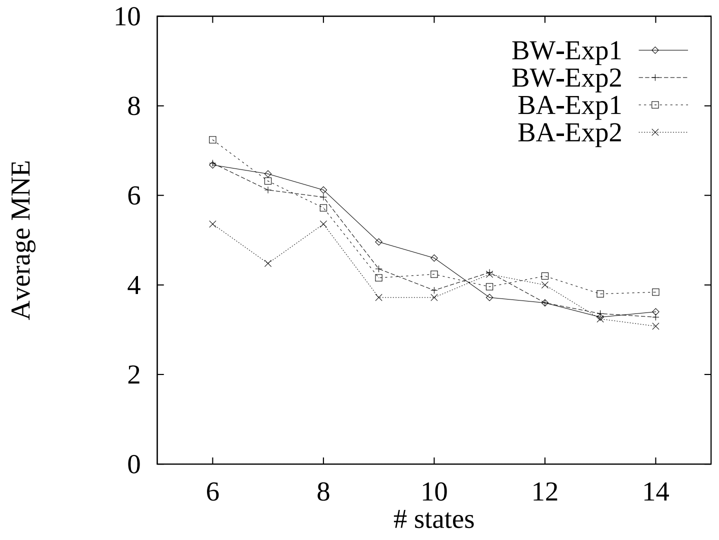


Figure 5.4: Average MNE for Exp1 and Exp2 in cross-validation test for lipocalin motif

only positive examples does not aim at discriminating between positive and negative examples, and thus in terms of the average MNE, the HMM first trained by MA turns into a worse model by being trained with BW or BA. On the other hand, the fact that the Exp2 results surpassed the BW and BA results indicates that choosing good priors is important for training HMMs in terms of sequence discrimination. From these results, we can clearly say that our method achieved higher discrimination ability than the conventional methods, especially in terms of the average MNE.

Finally, we show two HMMs which were trained by MA and whose MNE was zero. Figures 5.5 and 5.6 show the HMMs having nine and thirteen states respectively, and in these figures, we draw only the arcs whose state transition probabilities exceed 0.1 and the symbols whose output probabilities also exceed 0.1.

As shown in Figure 5.5, the HMM has only seven states at which symbols are emitted, whereas given training examples are twelve to fourteen residues long, and thus a transition path of an example must pass through some states twice or more. In other words, some of the seven states will likely play several different roles in given residues, and even in this situation, this HMM completely discriminates the positive and negative test examples. This result implies that from the viewpoint of sequence discrimination, we do not need to prepare a model in which the number of states is almost equal to the length of a given symbol sequence.

On the other hand, the HMM shown in Figure 5.6 has eleven states which output amino acids, and since this number is close to the length of a given training example, each position of the motif is expected to have its own state. Actually, in Figure 5.6, we can find the major transition path, which after once passing state 4 in the figure, returns to the state. In the path, only state 4 is used twice. It might be worth noting which positions in this motif are represented by state 4. Interestingly, one of the positions used twice is not the one shown in 'X' in the motif, but the one which is fixed at W. In general, to obtain HMMs with high sensitivity, any position which can be fixed at one amino acid in the motif ought to have its own state, but even though such a result is not obtained for this HMM, its MNE is equal to zero. One explanation for this result is that all negative examples used here have the motif, i.e. that any negative sequence has the W, and thus, the W was not essential to discriminate

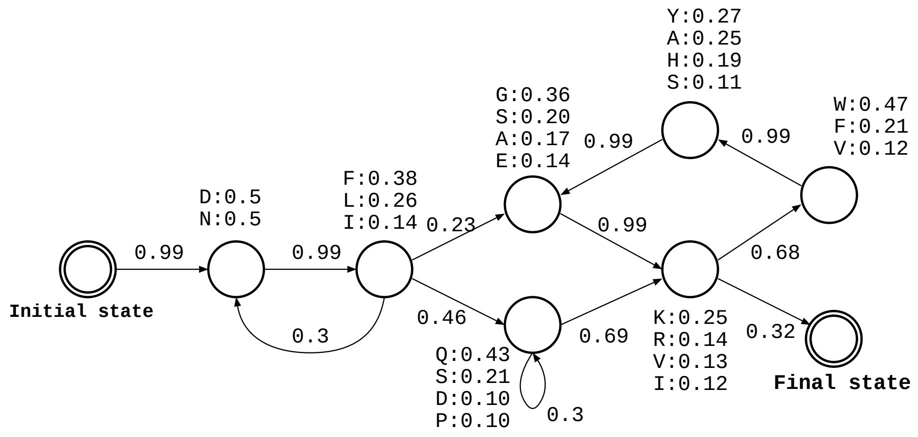


Figure 5.5: Hidden Markov model for lipocalin motif - 1

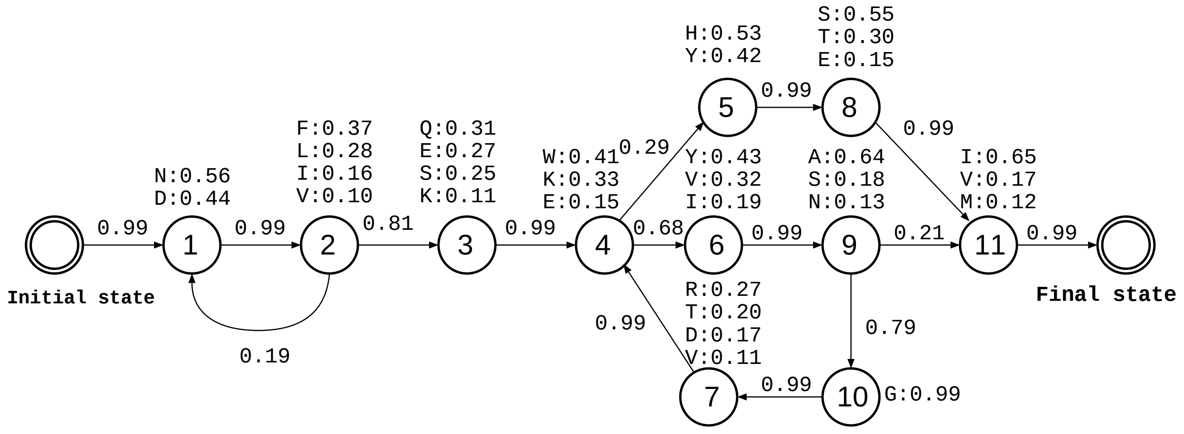


Figure 5.6: Hidden Markov model for lipocalin motif - 2

positive examples from negative ones in the data used here.

The major transition path shown in the figure displays two different types of paths after twice passing state 4. These two paths might imply that there exist two different types of correlations in training examples, though certainly this result is not verified experimentally. This case indicates that a fully-connected HMM can automatically provide this type of information which might be hidden in given training examples. This is another important advantage of the HMM method.

5.5 Experimental Results – 2

The binding of a major histocompatibility complex (MHC) molecule to a peptide originating in an antigen is essential to recognizing antigens in immune systems, and it has proven to be important to use computers to predict the peptides that will bind to an MHC molecule. In this section, first, we applied to this problem our supervised learning method of HMMs to test if our method surpasses existing methods for the problem of predicting MHC-binding peptides. Next, we generated peptides which are expected to have high probabilities to bind to a certain MHC molecule, called HLA-A2, based on our proposed method using peptides binding to HLA-A2 as a set of training data.

5.5.1 Data – MHC Binding Peptides

We obtained peptides and their ability to bind to MHC molecules (and activate T-cell proliferation) from the MHCPEP database developed by Brusic et al. (1997). Out of the 9,827 peptides in the current version of the database, there were 1,008 that are relevant to HLA-A2, which was the largest number among the peptides noted in the database.

In the MHCPEP database, there are two types of measures used in evaluating the ability of peptides, i.e. the ability to bind to MHC molecules (binding peptides) and to activate T-cell proliferation (activating peptides). Each peptide can be assigned one of six labels to indicate its ability: 'none (NO)', 'yes but little (YL)', 'yes but moderate (YM)', 'yes and high (YH)', 'yes but unknown' and 'unknown'. Out of the six labels, we use only four: NO, YL, YM and YH, because the peptides whose labels are unknown cannot be dealt with by our supervised learning method. Table 5.1 shows the number of peptides relevant to HLA-A2. The table shows all data obtained from the MHCPEP, and

	NO	YL	YM	YH	Total
binding	0	138	162	172	472
activating	79(53)	17(6)	46(13)	57(30)	199(102)

Table 5.1: Number of peptides relevant to HLA-A2 (Number of peptides of nine residues are shown in parentheses.)

thus any bias in the data is a result of the choice of peptides made by immunological experimenters. From the table, it can be seen that there are no peptides with 'NO' binding ability, and that the total number of binding peptides exceeds that of activating peptides in the table.

In order to represent the obtained peptide data, we used a fully-connected HMM, in which there is only one initial state and one final state, at which no symbol is outputted. In the fully-connected HMM, the initial state has arcs, each of which goes to any state except for the final state, and any other state except for the final state has arcs, each of which goes to any state other than the initial state. Here, let us describe how we set p^*_σ in our experiment. When the parameters of a fully-connected HMM of M states are assumed to be uniform distributions, i.e. $a_{ij} = \frac{1}{M}$ $(i, j = 1, \cdots, M)$ and $b_j(c) = \frac{1}{L}$ $(j = 1, \cdots, M, c = 1, \cdots, L)$, the likelihood that a peptide of length n is generated by the HMM is given as $\frac{1}{M}(\frac{1}{L})^n$ from Algorithm 2.5, where L is the number of types of symbols. In consideration of this calculation and based on a preliminary experiment, we fixed the target likelihood of a given peptide as $\mathcal{L}^{0.05}$ if it is in the YH class, $\mathcal{L}^{0.1}$ if it is YM, $\mathcal{L}^{0.2}$ if it is YL and $\mathcal{L}^{2.0}$ if it is NO, where $\mathcal{L} = \frac{1}{M}(\frac{1}{L})^n$.

Although the length of the obtained peptides shown in Table 5.1 ranges from seven to twenty-five, most of the peptides are nine to thirteen residues long. More concretely, in the binding peptides relevant to HLA-A2, 434 peptides (91.9% of the total) are nine to thirteen residues long, and in particular, the number of the peptides which are nine residues long is 192 and they occupy 61.9% of the total. Thus, we fixed the number of states of HMMs at twenty-two for HLA-A2, where the twenty-two states include an initial state and a final state, neither of which outputs any symbols. As the number of states was set at roughly twice the length of most peptides in training examples, the HMMs were expected to separately extract multiple sequence patterns hidden in the data.

5.5.2 Comparing Supervised Learning Method with Other Methods in Cross-validation Test

We compared the performance of our supervised learning algorithm of an HMM with those of two other methods. The first of the two was a back-propagation neural network, which had been used in predicting MHC binding peptides and regarded as the most effective approach. The second was the Baum-Welch algorithm, i.e. the most popular learning algorithm of an HMM. Using the Baum-Welch, we tested two-types of HMMs, i.e. fully-connected and alignment HMMs. A fully-connected HMM is one in which any pair of states is connected, except for the pair of the initial and final states. However, an alignment HMM has been used for aligning multiple sequences and (see Figure 2.14).

Data

In this experiment, testing was done by binary prediction, i.e. YES or NO. Thus, we used activating peptides relevant to HLA-A2, because there is no non-binding data in HLA-A2. Table 5.1 shows the number of peptides used in this experiment. Note that basically, there is no non-binding peptide in the MHCPEP database, because it gathers peptides that bind to MHC proteins. Thus, in the database, even the peptides which cannot activate T-cell proliferation will bind to MHC proteins. In other words, such peptides can be regarded as false-positive examples if a peptide which can activate T-cell proliferation is called a positive example. In this sense, the discrimination experiment performed here was a severe test. As back-propagation neural networks were used in the comparisons, all peptides used here were of nine residues, since all previous work based on back-propagation neural networks used only such peptides.

Note that the three methods differ in data usage. Our method used four labels, i.e. YH, YM, YL and NO. On the other hand, back-propagation neural networks were trained by two labels, i.e. YES or NO, as done in Gulukota et al. (1997), and the Baum-Welch algorithm used only YES (YH, YM, YL) since it is an unsupervised learning algorithm.

Back-propagation neural network

We here briefly explain the network used in our experiment, which is the same as the one used by Gulukota et al. (1997).

The network has three layers, i.e. an input, a hidden and an output layer, each of which consists of a fixed number of nodes. The number of input, hidden and output nodes is $180(= 20 \times 9), 50$ and 1, respectively. A set of 20 nodes in the input layer, each of which corresponds to one of 20 types of amino acids, corresponds to one of nine residues in a given peptide. When a peptide is given, only one node in the set of 20 nodes outputs 1 and the other 19 nodes in the set output 0. Any two nodes between input and hidden layers and between hidden and output layers are connected by a directed edge, to which a weight is attached.

Let x_j be the output value of node j and w_{ij} be the weight attached to the edge connecting from node i to node j. We calculate the x_j in the hidden and output layers as follows:

$$x_j = f(\sum_i w_{ij} x_i), \tag{5.9}$$

where the function f is a sigmoid function satisfying $f(x) = \frac{1}{1+e^{-x}}$. Weights w_{ij} are trained by a standard back-propagation learning algorithm (Rumelhart et al., 1986). In this learning, 1 is given as a teaching signal for the output value of this network if a given training peptide is labeled 'YES';

otherwise 0 is given, and the back-propagation minimizes the squared error-loss at the output node by a gradient descent algorithm.

In prediction, when a new peptide is given, the output value of the output node which can be calculated from (5.9), is given to the peptide.

Alignment HMM

As shown in Figure 2.14, an alignment HMM has a particular structure consisting of three types of states, i.e. matching (M1, M2, M3 in Figure 2.14), insertion (I0, I1, I2, I3 in the figure) and deletion (D1, D2, D3 in the figure) states. In the HMM, a matching state is a normal state which outputs a symbol according to a probability distribution attached to the state, while an insertion state emits a symbol according to a fixed uniform distribution and a deletion state does not emit any symbol.

In our experiment, the number of matching states was fixed at twenty, which is the same as the number of states in the fully-connected HMM used in the experiment, except for its initial and final states. The number of deletion and insertion states (twenty and twenty-one, respectively) is automatically determined from the number of matching states.

Experimental procedure

We randomly divide each set of peptides having a label into two, i.e. training and test, with a certain proportion of training data to all obtained data, and repeat this random division five times. That is, we generate five random sets of training and test data for a given proportion.

In training, we randomly generate five HMMs (or back-propagation neural networks) having different initial parameter values. For each of the five, we repeatedly train it and use it to predict unknown test data five times, with the five respective random sets of training and test data that was already generated. Thus, a total of twenty-five trials are done at a given proportion of training data to all data. We vary the proportion of training data to the whole data from 50% to 90% at 10% intervals.

In test, we measure the performance of each method by binary prediction, i.e. predicting whether a given peptide belongs to any of YH, YM and YL (i.e. YES) or NO. In this prediction, we consider the highest prediction accuracy (hereafter 'HPA') for test data which can be obtained by changing a cut-off value (which classifies test examples into two classes, i.e. YES and NO) for the output values of the test peptides. We calculate 25 HPAs for all 25 trials, and the performance of our method is evaluated by their average.

Learning curves

Figure 5.7 shows the learning curves of our supervised learning algorithm of HMMs and of back-propagation neural networks. As shown in the figure, the average HPA of the former was approximately 5–10% better than that of the latter at any proportion of training data to all data. From this result, we can say that in discriminating given new peptides, the performance of our supervised learning of HMMs exceeded that of a back-propagation neural network, which had been regarded as the most effective approach to this problem.

Figure 5.8 shows the learning curves of our supervised learning algorithm of fully-connected HMMs and the Baum-Welch algorithm of fully-connected or alignment HMMs. This figure indicates that the fully-connected HMM was able to greatly improve the average HPA obtained by the alignment HMM, and that our supervised learning was able to further improve the HPA obtained by the Baum-Welch. The average HPA of fully-connected HMMs trained by our method was always approximately 2–15%

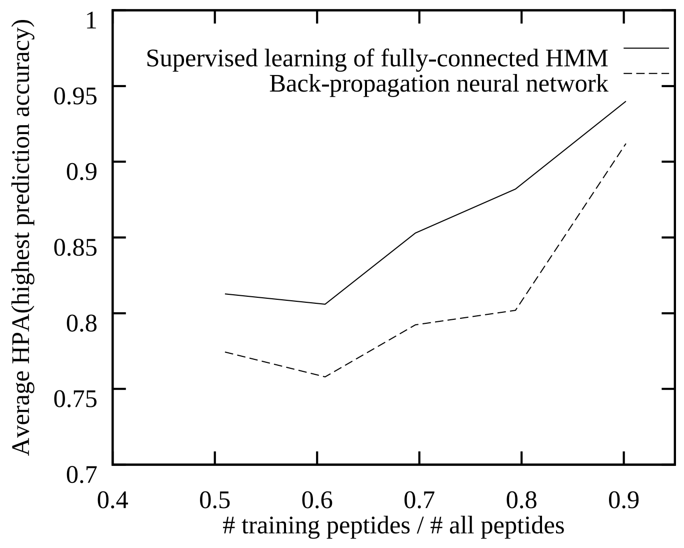


Figure 5.7: Comparing our supervised learning of hidden Markov models with back-propagation neural networks

better than those of fully-connected and alignment HMMs trained by the Baum-Welch.

Figures 5.7 and 5.8 clearly demonstrate that our method surpasses all of the methods used for comparison purposes.

5.5.3 Predicting Peptides That Bind to MHC Molecule

Data

The number of peptides relevant to activity is considerably smaller than that for binding to HLA-A2, and hence, we here consider binding peptides only. Thus, the data used here has only three types of labels, i.e. YH, YM and YL, and is a set of positive examples, which can bind to HLA-A2.

Experimental procedure

We train an HMM by our supervised learning algorithm using all data of HLA-A2, and repeat this training 100 times with random different initial parameter values. Out of the 100 trained HMMs, we choose the one which provides the minimum value of error-distance function F (see (5.4)) for all training data of peptides which can bind to HLA-A2.

Next, we perform a random walk on the chosen HMM trained by peptides that bind to HLA-A2. We start at the initial state of the HMM, and randomly choose a state to transit depending on the transition probabilities attached to the arcs from the initial state, and after moving to a state, we again randomly choose a symbol depending on the symbol output probability attached to the state. We repeat this state transition and symbol outputting until the transition reaches the final state. This random walk finally generates a sequence and the score of the sequence, which is obtained by multiplying all the probabilities used for generating the sequence on state transition and outputting a symbol of the walk. Roughly speaking, we can regard the score as the likelihood that the sequence is generated by the HMM, as the Viterbi algorithm is used in predicting the likelihood.

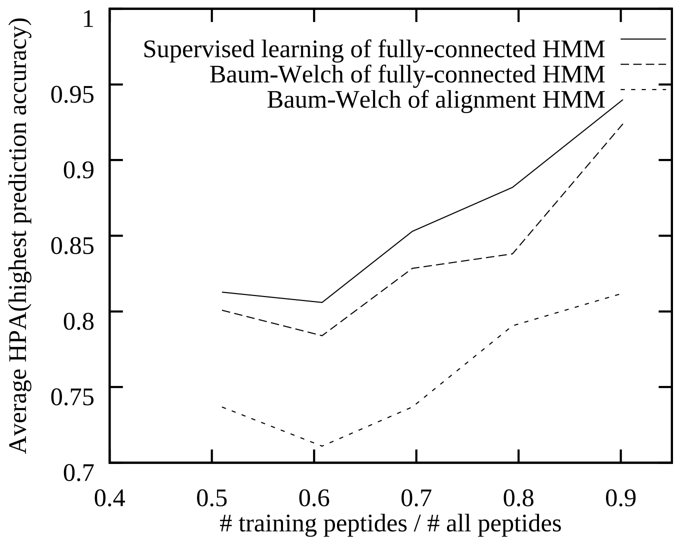


Figure 5.8: Comparing our supervised learning with Baum-Welch

We repeat the random walks 100,000 times, and out of the 100,000 sequences generated, we remove those which have already been noted in the MHCPEP database and those which consist of only one type of amino acid. Out of the sequences generated, we extract only the sequences which are nine residues long, because as mentioned earlier such peptides of nine residues occupy more than 60% of all the peptides relevant to HLA-A2. We sort the processed sequences in the descending order of their scores, and select 10,000 of them from the top down.

Finally, we repeat the above process five times. We sort the obtained five sets of 10,000 in the descending order of their scores, and select the top 100 of them.

Results

Figure 5.9 shows the HMM which provides the lowest value of error-distance function F for all training peptides that bind to HLA-A2 protein, in 100 HMMs trained by our supervised learning algorithm using the same data. In the figure, only arcs whose transition probabilities exceed 0.1 and the top three symbols (at maximum) whose symbol output probabilities exceed 0.05 at each state, are shown. The arc having the largest transition probability of the probabilities attached to arcs starting from a state is shown by a thick line, and states are numbered 1 to 20 from left to right and from top to bottom.

Note that the HMM automatically extracts roughly two different patterns hidden in the peptides used as training data. One major pattern is states $1 \to 2 \to 4 \to 7 \to 9 \to 12 \to 14 \to 17 \to 20$, and the other relatively minor pattern is states $1 \to 2 \to 3 \to 5 \to 8 \to 11 \to 13 \to ((18(\to 19 \to 20))$ or $(16 \to 20))$. As shown in the figure, a number of variations can be incorporated in the second pattern, but no change is allowed in the first pattern.

States 2 and 20, which can be used in the two patterns, coincide with two residues of a motif of HLA-A2 reported by Falk et al. (1991). They are L at position 2 and V at position 9, and these have high probabilities at states 2 and 20, respectively.

In the first pattern, all states except for states 2 and 20, have a broad distribution of symbol

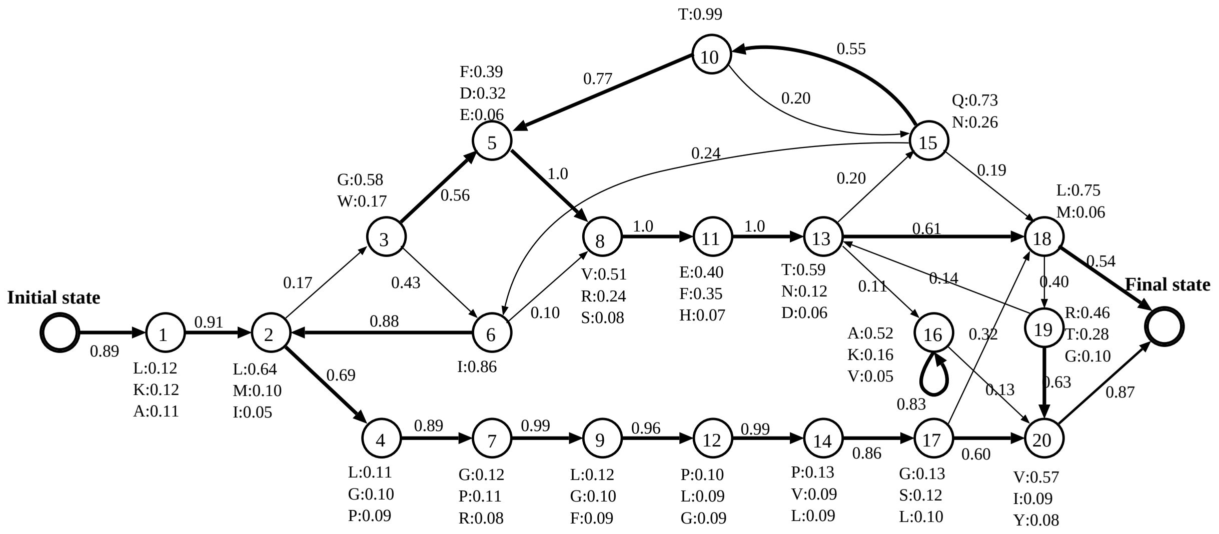


Figure 5.9: Main part of HMM representing peptides binding to HLA-A2

output probabilities, in which the largest probability value is at maximum 0.13. Furthermore, such distributions at states 4, 7, 9, 12, and 14 are similar to each other, and in them, G, L and P always have relatively high probabilities. This result is consistent with a report by Sette et al. (1991), in which positions 3 to 5 in a HLA-A2 motif have the same amino acid propensity. This indicates, however, that neither the motif nor the first pattern can capture any distinct feature of this portion, and thus it will be difficult for them to accurately predict (or discriminate) peptides that bind to HLA-A2 protein.

On the other hand, the second pattern presents a more clear sequence pattern hidden in the training data. In particular, the transition of states $5 \rightarrow 8 \rightarrow 11 \rightarrow 13$ is connected by arcs, at any of which a transition probability of 1.0 is attached, and this indicates that the transition is certainly hidden in training peptide data. Actually, an MHC binding peptide experimentally found in Influenza matrix protein (Gotch et al., 1988) contains the amino acid sequence FVFT, which can be generated with a high probability by this transition. The sequence is found in 49 of the total 472 peptides used as training data, and this is one of the most frequent patterns in HLA-A2 binding peptides. Note that the sequence FVFT is found in a different position in each of the 49 training peptides. Out of the 49 peptides, the number in which the sequence starts at the fourth, fifth, sixth, seventh and eighth position is 3, 23, 20, 2 and 1, respectively.

We can find other frequent sequence patterns in the second pattern. For example, the longer sequence LGFVFT, which can be generated by states $2 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 11 \rightarrow 13$ with a high probability, is found in 36 peptides in the training data. Similarly, the sequence TLTV, which can be generated by states $13 \rightarrow 18 \rightarrow 19 \rightarrow 20$, is in 33 peptides in training data. All frequent sequence patterns, which are revealed by the HMM of Figure 5.9, are shown in Table 5.2. The patterns presented in the table are those which are longer than three and which are found in more than 30 peptides in the training data. If longer patterns, including those which satisfy the above requirement, are found in more than 30 peptides, only the longest one of them is described. The table indicates that each portion of the second pattern in the HMM captures hidden features in the training data.

Table 5.3 shows the top 100 peptides which were obtained by our random generation process

Patterns	State transition	# peptides
GILGF	$3 \rightarrow 6 \rightarrow 2 \rightarrow 3 \rightarrow 5$	33
ILGFVF	$6 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 11$	34
LGFVFT	$2 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 11 \rightarrow 13$	36
GFVFTL	$3 \rightarrow 5 \rightarrow 8 \rightarrow 11 \rightarrow 13 \rightarrow 18$	36
FTLTV	$11 \rightarrow 13 \rightarrow 18 \rightarrow 19 \rightarrow 20$	30

Table 5.2: Patterns which can be generated by HMM in Figure 5.9 with high probabilities and which are frequently seen in peptides of training data

described above and whose binding ability had been so far unknown. Most of the obtained 100 peptides have the pattern which is supposed to be generated by states $5 \rightarrow 8 \rightarrow 11 \rightarrow 13$, and this indicates that most of the generated peptides belong to the second pattern presented by the HMM of Figure 5.9.

5.6 Conclusion

We established a new learning method for training hidden Markov models (HMMs) for discriminating unknown sequences. Our method allows us to learn an HMM using data consisting of continuous or discrete classes or data having real-valued labels. In brief, our method allows us to perform supervised learning in place of unsupervised learning which is done in the Baum-Welch and Baldi algorithms. In our learning method, to make this feature possible, we set an error-distance function, which measures an error-distance between the real likelihoods and the target likelihoods for given training sequences, and use a gradient descent method so that the function should be smoothly minimized. Experimental results show that in a cross-validation test, our method made fewer discrimination errors than the conventional Baum-Welch and Baldi methods for HMMs and a neural network learning method. In addition, the computation time required to train HMMs by our method is on the same order as that in the two conventional learning methods for HMMs. From these results, we conclude that the new supervised learning strategy we established provides a greater discrimination accuracy than that provided by the methods which are considered state-of-the-art in this field.

Furthermore, we should point out that our method can be applicable to any stochastic model for which the training algorithms are based on the Baum-Welch learning. Specifically, stochastic context-free grammars (Sakakibara et al., 1994; Eddy & Durbin, 1994; Grate, 1995; Durbin et al., 1998) and the stochastic tree grammars presented in Chapter 6, which have been applied to both the natural language processing and computational molecular biology field, are trained by the Inside-Outside algorithm corresponding to the extended Baum-Welch method. Our supervised learning method also can be expanded to train these grammars.

	Peptide	Logarithmic likelihood
1	GILGFVETL	-5.13036
2	GILGDVETL	-5.21152
3	GILGDVFTL	-5.26656
4	LGFVETLRV	-5.47421
5	KGFVETLRV	-5.47863
6	GILGFRFTL	-5.50243
7	GILGDRETL	-5.52856
8	KGFVFTLRV	-5.53366
9	LGDVETLRV	-5.55537
10	KGDVETLRV	-5.55979
11	GILGDRFTL	-5.58359
12	GFVETLRTL	-5.58546
13	LGDVFTLRV	-5.61040
14	KGDVFTLRV	-5.61482
15	GFVFTLRTL	-5.64049
16	GGFVETLRV	-5.64502
17	AGDVFTLRV	-5.68058
18	LGFVETLTV	-5.69142
19	KGFVETLTV	-5.69584
20	GGFVFTLRV	-5.70005
21	GILWFVFTL	-5.71087
22	GDVFTLRTL	-5.72165
23	GGDVETLRV	-5.72618
24	KLGFVETQL	-5.73604
25	GILWDVETL	-5.73699
26	WILGDVETL	-5.73699
27	GFVETQLRV	-5.74338
28	FGFVETLRV	-5.74811
29	KGFVFTLTV	-5.75087
30	AGFVETLTV	-5.76159
31	IGFVETLRV	-5.77005
32	LGDVETLTV	-5.77257
33	GGDVFTLRV	-5.78121
34	KLGFVFTQL	-5.79107
35	LGFRETLRV	-5.79124
36	GILWDVFTL	-5.79203
37	KGFRETLRV	-5.79566
38	ALGFVETQL	-5.80179
39	FGFVFTLRV	-5.80315
40	LLGFVETLL	-5.80687
41	KLGFVETLL	-5.81129
42	LLGDVETQL	-5.81278
43	AGFVFTLTV	-5.81663
44	KLGDVETQL	-5.81720
45	GDVETQLRV	-5.82453
46	LGDVFTLTV	-5.82761
47	KGDVFTLTV	-5.83203
48	FLAAAAAAV	-5.85104
49	IGDVETLRV	-5.85121
50	GFVFTLTTL	-5.85770
51	AGFRETLRV	-5.86142
52	GGFVETLTV	-5.86222
53	KLGFVFTLL	-5.86632
54	LLGDVFTQL	-5.86781
55	GILGFVFSL	-5.87013
56	GFVFTLLRV	-5.87366
57	GDVFTQLRV	-5.87957
58	FGDVFTLRV	-5.88431
59	LLGDVETLL	-5.88803
60	LLGFVFTRV	-5.88970
61	GILGDVESL	-5.89625
62	AGDVFTLTV	-5.89778
63	LLLGFVETL	-5.89835
64	GDVETLLRV	-5.89978
65	GFRETLRTL	-5.90249
66	ALGFVETRV	-5.90484
67	IGDVFTLRV	-5.90625
68	GILGDVENL	-5.90660
69	GGFVFTLTV	-5.91726
70	KLGDVETRV	-5.92025
71	GDVETQTQL	-5.94344
72	RGFVETLRV	-5.94488
73	GILAAAAAV	-5.94896
74	GIMGFVETL	-5.95025
75	TGDVETLRV	-5.95214
76	GLGFVFTQL	-5.95746
77	ALGDVETLL	-5.95820
78	GFVETQLTV	-5.96058
79	GILGDVFNL	-5.96164
80	GGFRETLRV	-5.96205
81	FGFVETLTV	-5.96532
82	LLGDVFTRV	-5.97086
83	KLGDVFTRV	-5.97528
84	GILGEVETL	-5.97586
85	GFVETLRVL	-5.97601
86	GLGFVETLL	-5.97768
87	LLLGDVETL	-5.97951
88	YLAAAAAAV	-5.98183
89	GDRETLRTL	-5.98365
90	KLLGDVETL	-5.98393
91	ALGDVETRV	-5.98600
92	GILGDVHTL	-5.99664
93	GGDVFTLTV	-5.99842
94	GDVFTQTQL	-5.99848
95	LWFVETLRV	-5.99968
96	RGFVFTLRV	-5.99992
97	KWFVETLRV	-6.00410
98	FLGFVETQL	-6.00552
99	GFVFTQLTV	-6.01562
100	CLAAAAAAV	-6.02506

Table 5.3: Top 100 peptides of nine residues that bind to HLA-A2 protein, generated by HMM of Figure 5.9

Chapter 6

Discovering Common β-sheets with Stochastic Tree Grammar Learning

6.1 Introduction

Protein is made of a sequence of amino acids which is folded into a 3-dimensional structure. Protein secondary structures are relatively small groups of protein structures exhibiting certain notable and regular characteristics, which function as intermediate building blocks of the overall protein structure, and can be classified into three types; α-helix, β-sheet, and others. We established a new method for predicting protein secondary structure of a given amino acid sequence, based on a classification rule automatically learned from a relatively small database of sequences whose secondary structures have been experimentally determined (Mamitsuka & Abe, 1994a; Abe & Mamitsuka, 1997, 1994). Among the above three types, we concentrated on the problem of predicting β-sheet regions in a given amino acid sequence. Our strategy receives as input amino acid sequences known to contain β-sheet regions, and train the probability parameters of a certain type of stochastic tree grammar so that its distribution best approximates the patterns of the input sample. Some of the rules in the grammar are intended a priori for generating β-sheet regions and others for non-β-sheets. After training, the method is given a sequence of amino acids with unknown secondary structure, and predicts according to which regions are generated by the β-sheet rules, in the most likely parse for the input sequence.

The problem of predicting protein structures from their amino acid sequences is probably the single most important problem in computational molecular biology with immense scientific significance and broad engineering applications. Since the early 1990s, increasing attention has been given to the three-dimensional structural prediction methods which attempt to predict the entire protein structure, such as homology modeling (c.f. Sánchez & Sali, 1997) and remote homology modeling (c.f. Jones, 1997). These methods are based on alignment/scoring of the test sequence against sequences with known structure, and therefore are not effective for those sequences having less than 25% sequence similarity to the training sequences (Sander & Schneider, 1991). At the other end of the spectrum is the protein secondary structure prediction approach, which is general in the sense that it does not rely on alignment with sequences with known structure and hence can be applied on sequences having little or no sequence similarity, but provides less information. The present chapter aims at providing a general method which can be applied to sequences with less than 25% sequence similarity, and yet provides more structural information.

The classical secondary structure prediction problem is the problem of determining which regions in a given amino acid sequence correspond to each of the above three categories (Barton, 1995; Cuff & Barton, 1999). At the midst of 1990s, there were several approaches for the prediction of α-helix regions using machine learning techniques which achieved moderate success. The prediction rates of approximate 80 percent, varying depending on the exact conditions of experimentation, achieved by some of these methods (Mamitsuka & Yamanishi, 1992, 1995; Rost & Sander, 1993b). The problem of predicting β-sheet regions, however, was not treated at a comparable level. This asymmetry can be attributed to the property of β-sheets that their structures typically range over several discontinuous sections in an amino acid sequence, whereas the structures of α-helix are continuous and their dependency patterns are more regular.

To cope with this difficulty, we defined a certain family of stochastic tree grammars whose expressive powers exceed not only that of hidden Markov models (HMMs), but also stochastic context free grammars (SCFGs). SCFGs are used extensively in speech recognition, and have been introduced to computational molecular biology (Sakakibara et al., 1994; Durbin et al., 1998). Context free grammars are not powerful enough to capture the kind of long-distance dependencies exhibited by the amino acid sequences of β-sheet regions. This is because the β-sheet regions exhibit both the 'anti-parallel' dependency (of the type *abccba*), and 'parallel' dependency (of the type *abcabc*), and moreover, various combinations of them (as, for example, in *abccbaabcabc*). The class of stochastic tree grammars which we will define in this chapter, the *stochastic ranked node rewriting grammar* (SRNRG), is one of the rare families of grammatical systems that have both enough expressive power to cope with all of these dependencies and at the same time enjoy relatively efficient parsability and learnability. Searls (1993) noted that the language of β-sheets is beyond context free and suggested that they are indexed languages. Indexed languages are not recognizable in polynomial time, however, and hence indexed grammars are not useful for our purpose. RNRG falls between them and appears to be just what we need.

The ranked node rewriting grammars (RNRGs) were briefly introduced in the context of computationally efficient learnability of grammars by Abe (Abe, 1988), but its formal properties as well as basic methods such as parsing and learning algorithms were left for future research. The discovery of RNRGs was inspired by the pioneering work of Joshi et al (see e.g. Joshi, Levy, & Takahashi, 1975; Vijay-Shanker & Joshi, 1985) on a tree grammatical system for natural language called 'Tree Adjoining Grammars' (or TAGs for short), but RNRGs generalizes TAGs just in a way that is suited to capture the type of dependencies present in the sequences in β-sheet regions. For example, TAG can handle a single parallel dependency, which cannot be handled by CFG, but cannot deal with a complicated combination of anti-parallel and parallel dependencies. All of such dependencies can be captured by some members of the RNRG family.

The learning algorithm we established is an extension of the 'Inside-Outside' algorithm for the stochastic context free grammars (Jelinik, Lafferty, & Mercer, 1990), and is also related to the extension of the Inside-Outside algorithm developed for the stochastic tree adjoining grammars by Schabes (1992). These are extended versions of the Baum-Welch algorithm for HMMs and all iterative algorithms guaranteed to find a local optimal for the maximum likelihood settings of the rule application probabilities. Perhaps the most serious difficulty with our method is the extensive computation required by the parsing and learning algorithms. The computational requirement of the learning algorithm is brought down drastically by using the so-called 'bracketing' technique. That is, when training an SRNRG fragment corresponding to a certain β-sheet region, rather than feeding the entire input sequence to the learning algorithm, the concatenation of just those substrings of the input sequence which correspond to the β-sheet region is fed to the learning algorithm. In contrast, the parsing algorithm must process the entire input string, as it clearly does not know in advance which substrings in the input sequence correspond to the β-sheet region. Hence most of the computational requirement is concentrated on the parsing algorithm.

In order to reduce the computational requirement of the parsing algorithm, we restricted the form of grammars to a certain subclass of SRNRG which we call linear SRNRG and devised a simpler and faster algorithm for the subclass. This subclass is obtained by placing the restriction that the right hand side of any rewriting rule contains exactly one occurrence of a non-terminal, excepting lexical non-terminals. This significantly simplifies the parsing and learning algorithms as the computation of each entry in the parsing table becomes much simpler. In addition, we parallelized our parsing algorithm and implemented it on a 32-processor CM-5 parallel machine. We were able to obtain a nearly linear speed-up, and were able to predict the structure of a test sequence of length 100 in about a minute, whereas the same task took our sequential algorithm almost 30 minutes.

We also established a method of reducing the alphabet size (i.e. the number of amino acids) by clustering them using MDL (Minimum Description Length) approximation (Rissanen, 1989) and their physico-chemical properties, gradually through the iterations of the learning algorithm. The physico-chemical properties we used are the molecular weight and the hydrophobicity, which were used in Chapter 3 for predicting α-helix regions. That is, after each iteration of the learning algorithm, we attempted to merge a small number of amino acids, if doing so reduces the total description length using the likelihood calculation performed up to that point.

As a preliminary experiment, we first used three toxins which have a common 3-dimensional structure shown in Figure 6.4, but each of three has less than 25% pairwise sequence similarity. The results obtained indicate that our method is able to capture and generalize the type of long-distance dependencies that characterize β-sheets of the toxins. Using an SRNRG trained by data for one of three toxins, our method was actually able to predict the location and structure of β-sheets in the other two toxins. With the training sequences and the test sequences having less than 25% sequence identity, such a prediction problem belongs to what is sometimes referred to in the literature as the 'Twilight Zone' (Doolittle et al., 1986), where alignment is no longer effective.

Then, we conducted a large-scale experiment in which proteins used for training and those for testing not only have almost no sequence similarity, but their structures have no apparent relationship. In the experiment, we restricted our attention on the β-sheet regions that can be expressed using rank 1 linear SRNRGs, which we call *rank 1 four-strand patterns* or *rank 1 four-strand β-sheets*. Furthermore, for simplicity, we write *four-strand patterns* for rank 1 four-strand patterns. As training data we used all the sequences (satisfying a certain weak condition), listed in PDB_SELECT 25% list (Hoboem et al., 1992), a database containing sequences of protein with known structure possessing at most 25% sequence similarity with one another. For each one of these sequences, we enhanced it by the set of aligned sequences listed in the HSSP (Homology-derived Secondary Structure of Proteins) (Sander & Schneider, 1991) for it, as is done by a number of secondary structure prediction methods (Rost & Sander, 1993a; Mamitsuka & Yamanishi, 1995). As test sequences, we took all sequences in PDB_SELECT 25% list of length at most 200, excluding the training sequences themselves, containing relevant types of four-strand patterns. The above process translates to saying that we used essentially all data that are currently available, and hence without any bias. The prediction was made according to the location and structure of the most likely analysis given by these grammar fragments, except the fragment trained on sequences that have less than 25% sequence similarity to the test sequence, thereby ensuring that the prediction could not be done easily by homology modeling.

The results of our experiment indicate that the prediction made by our method on the approximate locations and structures of arbitrary rank 1 four-strand β-sheets is statistically significant, although its predictive accuracy is not yet at a satisfactory level. Our method was able to actually predict the structure of a couple of rank 1 four-strand β-sheets approximately correctly, based on stochastic grammar fragments trained on sequences of proteins that are very different from that of the test sequence, thus discovering a hitherto unnoticed similarity that exists between local structures of apparently unrelated proteins. Also, in the course of the experiments, it was observed that the prediction is much easier when we restrict the test sequences to contain relatively isolated β-sheets, and not partial β-sheets existing as part of a larger β-sheet structure. Interestingly, it was found that for prediction of these partial β-sheet structures, training data from relatively isolated β-sheets was not only useless but even harmful, in the sense that including them as part of the training data actually degrades the predictive performance. These observations together suggest that: (i) There exist some similarities between the sequences of relatively isolated four-strand patterns in different proteins, and acquiring generalized patterns for them can help improve prediction accuracy. (ii) The sequences of four-strand patterns in larger β-sheet structures are considerably different from those of relatively isolated four-strand patterns, and (iii) Satisfactory prediction of larger β-sheet structures would probably require more global information than the level of four-strand patterns.

As a final experiment, we examined how each of the training/test sequences used in our experiment is classified in an existing protein classification, in particular the protein classification database SCOP (Murzin et al., 1995). We observed that in cases where the β-sheet structure of a test sequence is relatively accurately predicted, the training sequences were not classified in the same family or superfamily with the test sequences. This observation suggests the possibility that the classification induced by the domains of cross-predictability is different from existing classifications, and in particular, consists of significantly fewer clusters. Viewed differently, this fact suggests that our prediction method is potentially useful as a tool for scientific discovery, so to speak, of local structures that are commonly shared by proteins that are non-homologous and are classified differently in any existing classification.

A brief bit of information relating to the work described in this chapter: Since we proposed our method, some secondary structure prediction methods have been reported which provide structural information such as that of β-strand 'contacts' between pairs of long distance residues (Hubbard & Park, 1995; Riis & Krogh, 1996) and which are based on the same motivation as that of our method. These methods (as well as our method) successfully predicted some residue contacts in β-sheets, but their disadvantage lies in the fact that they do not have a systematic formalism like our SRNRGs. In addition, we emphasize that, unlike them, our method is able to predict not only the residue contacts but also the whole structure of a β-sheet for a test sequence. Since we proposed our method, no group except one has used tree grammars in the field of computational biology, because of their rather high computational complexity. The group which proposed a tree-grammar based method used TAGs and applied them to predicting the structure of RNA sequences (Umemura et al., 1999), but in them, a fixed grammar was used and no learning algorithm for the grammar was presented by them. We emphasize that our method allows us to learn the probability parameters of our grammar from given examples and cut down on the high computational complexity of tree grammars with a parallelized parsing algorithm.

6.2 Stochastic Tree Grammars and β-Sheet Structures

6.2.1 Stochastic Ranked Node Rewriting Grammars (SRNRGs)

We recall the definition of ranked node rewriting grammar (RNRG), which is given in Chapter 2.

Definition 6.1 (Ranked Node Rewriting Grammar) (Abe & Mamitsuka, 1997) A *ranked node rewriting grammar* G is a 5-tuple $\langle \Sigma_N, \Sigma_T, \sharp, \beta_G, R_G \rangle$, where:
(i) Σ_N is a ranked alphabet called the *non-terminal alphabet* of G.
(ii) Σ_T is a ranked alphabet such that each symbol in Σ_T has rank 0 and is disjoint from Σ_N. The alphabet Σ_T is called the *terminal alphabet* of G.
(iii) $\sharp$ is a distinguished symbol distinct from any member of $\Sigma_N \cup \Sigma_T$, indicating an *empty node*. Let

$\Lambda = \Sigma_N \cup \Sigma_T \cup \{\sharp, \lambda\} \cup \{t_i | i = 1, 2, \cdots\}$ be a ranked alphabet, where λ is the empty string, both $\sharp$ and λ have rank 0 and the rank of t_i is i for $i \geq 1$. For convenience, we confuse t_i's and denote them simply by t.
(iv) β_G is a tree in T_Λ such that no node is labeled with $\sharp$. We call β_G the *starting tree* of G.
(v) Let α be a tree in T_Λ. We define the *rank of* α, denoted by $rank(\alpha)$, the number of nodes in α labeled with $\sharp$. A tree α with $rank(\alpha) \geq 1$ is called an *incomplete tree.* A *rewriting rule* of G is a pair $\langle S, \alpha \rangle$ such that S is a non-terminal symbol in Σ_N and α is a tree in T_Λ with $rank(S) = rank(\alpha)$. We write $S \rightarrow \alpha$ for the rewriting rule $\langle S, \alpha \rangle$. Then R_G is a finite set of rewriting rules of G.
We define the *rank of* G by $rank(G) = \max \{rank(S) \mid S \in \Sigma_N\}$. If $k = rank(G)$, then G is called a *rank k grammar.* □

We emphasize that the distinction between non-terminal symbols and terminal symbols described above does not coincide with that between internal nodes and frontier nodes. See an example derivation given in Figure 2.15 (b). In the figure, t is an internal node, and a and b are frontier nodes.

Definition 6.2 (Derivation) (Abe & Mamitsuka, 1997) Let $G = \langle \Sigma_N, \Sigma_T, \sharp, \beta_G, R_G \rangle$ be an RNRG. We say G *derives A from B in one step* if there is a rewriting rule $r = \langle S_r, \alpha_r \rangle$ such that some node η of A is labeled by S_r, and B is the tree obtained by replacing η in A by α_r. We write $A \vdash_G B$ and let $\vdash_G^*$ denote the transitive closure of the relation $\vdash_G$. β_G and β obtained by $\beta_G \vdash_G^* \beta$ are called *partially derived trees.* □

The tree language of a grammar is defined as the set of trees over the terminal alphabet, which can be derived from the grammar. This is analogous to the way the string language of a rewriting grammar is defined in the Chomsky hierarchy.

Definition 6.3 (Tree Language and String Language) (Abe & Mamitsuka, 1997)
Let $G = \langle \Sigma_N, \Sigma_T, \sharp, \beta_G, R_G \rangle$ be an RNRG. The *tree language* $T(G)$ and *string language* $L(G)$ of G are defined as follows:

$$\begin{aligned} T(G) &= \{\beta \in T_{\Sigma_T} \mid \beta_G \vdash_G^* \beta\}, \\ L(G) &= \{\text{yield}(\beta) \mid \beta \in T(G)\}, \end{aligned}$$

where yield(β) is the strings that appear on the leaves of β, read from left to right. □

If we place an upper bound, say k, on the rank of a node that can be rewritten, we obtain a family of grammars, RNRG(k), each of which has varying expressive power.

Definition 6.4 (RNRG Hierarchy) (Abe & Mamitsuka, 1997) Let $G = \langle \Sigma_N, \Sigma_T, \sharp, \beta_G, R_G \rangle$ be an RNRG. For each $k \in N$, we let RNRG(k) $= \{G \in \text{RNRG} \mid rank(G) \leq k\}$. For each $k \in N$, we let RNRL(k) $= \{L(G) | G \in \text{RNRG}(k)\}$. □

In Definition 6.4, one can easily verify that RNRL(0) equals the class of context free languages (CFL), and RNRL(1) equals the class of tree adjoining languages (TAL). Furthermore, for any $k \geq 2$, RNRL(k) contains the $2(k+1)$ count language, namely the language $\{a_1^n a_2^n \cdots a_{2(k+1)}^n | n \in N\}$, but RNRL($k-1$) does not. We now give some examples of RNRG grammars. The language $L_1 = \{ww^Rww^R | w \in \{a, b\}\}$ is generated by the RNRG(1) grammar G_1 shown in Figure 2.15 (a), where w denotes a string, i.e. a word, and w^R denotes the string obtained by 'reversing' w.

Figure 2.15 (b) shows the derivation of the string *abbaabba* by G_1. L_1 can be generated by a tree adjoining grammar, but the '3 copy' language, $L_2 = \{www \mid w \in \{a, b\}^*\}$ cannot (c.f. Vijay-Shanker

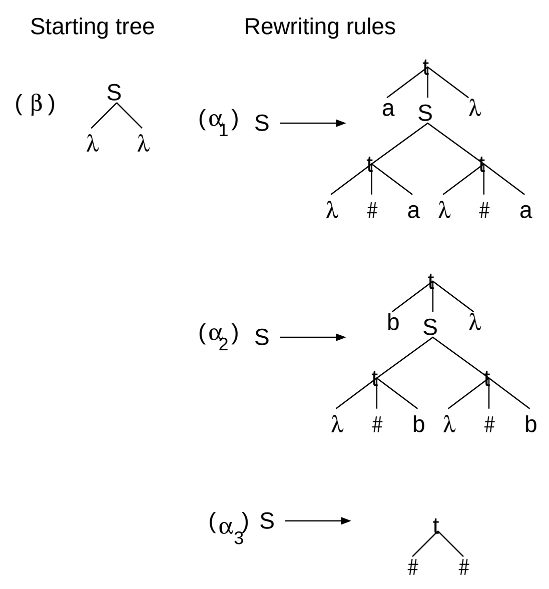


Figure 6.1: An RNRG(2) grammar G_2 generating $\{www|w \in \{a,b\}^*\}$

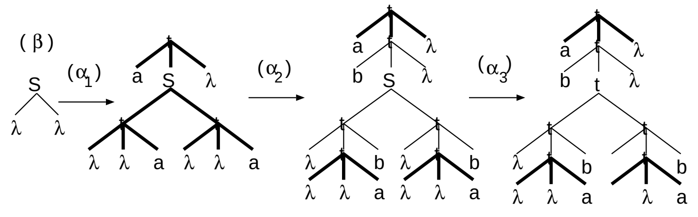


Figure 6.2: Derivation of 'ababab' by an RNRG-2 grammar

& Joshi, 1985). This language can be generated by the RNRG(2) grammar G_2 shown in Figure 6.1. Figure 6.2 shows the derivation of the string *ababab* by G_2. As shown in the figure, note that the tree structure introduced by a particular rewriting rule may be split into several pieces in the final derived tree, unlike usual parse trees in CFG. It is easy to see in the figure that the three occurrences of letter a are generated by a single occurrence of the rewriting rule (α_1) and its 'cross-serial' dependency therefore is captured by a single rule.

Given the definition of RNRG, the stochastic RNRG is defined analogously to the way stochastic CFG is defined from CFG. That is, associated with each rewriting rule in a stochastic RNRG is its rule application probability, which is constrained so that for each non-terminal, the sum total of rule application probabilities of all the rewriting rules for that non-terminal equals unity. We recall the definition of stochastic ranked node rewriting grammar (SRNRG), which is given in Chapter 2.

Definition 6.5 (Stochastic Ranked Node Rewriting Grammar) (Abe & Mamitsuka, 1997) A *stochastic ranked node rewriting grammar* G is a 6-tuple $\langle \Sigma_N, \Sigma_T, \sharp, \beta_G, R_G, T_G \rangle$, where:
(i) The 5-tuple $\langle \Sigma_N, \Sigma_T, \sharp, \beta_G, R_G \rangle$ is an RNRG.
(ii) For a rewriting rule $r = \langle S_r, \alpha_r \rangle$, $T_G(r)$ is a *rule application probability* of r, where $\sum_{\{r|S_r=S\}} T_G(r) =$

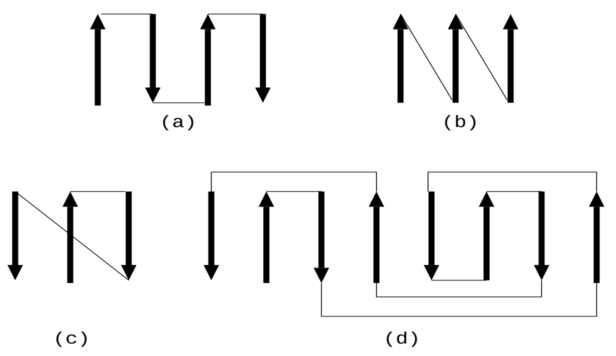


Figure 6.3: Some typical β-sheet structures

1 is assumed for each $S \in \Sigma_N$. $\square$

This way, each stochastic RNRG can be viewed as a probabilistic generator of finite strings, and defines a probability distribution over the set of all finite strings. For example, if we assign the probabilities 0.5, 0.3, and 0.2 to the three rewriting rules α_1, α_2, and α_3 in G_2, then it generates the string *ababab* with probability $0.5 \times 0.3 \times 0.2 = 0.03$.

6.2.2 Modeling β-Sheet Structures with RNRG

As we noted in Section 6.1, the prediction of the β-sheet regions has been considered difficult, because of the long distance dependencies exhibited by the corresponding regions. This is illustrated by the schematic representation of typical β-sheet structures given in Figure 6.3.

The arrows indicate the β-strands, and the line going through them is the amino acid sequence. The β-sheet structure is retained by hydrogen bonds (H-bonds) between the corresponding amino acids in neighbouring strands, so it is reasonable to suspect that there are correlations between the amino acids in those positions. The structure exhibited in Figure 6.3 (a) is known as the anti-parallel β-sheet, as the dependency follows the pattern *..abc..cba..abc..cba..*, where the use of an identical letter indicates that those positions are connected by H-bonds and believed to be correlated. Note that to be more precise, one out of every two is connected by an H-bond. In contrast, the structure exhibited in Figure 6.3 (b) is known as the parallel β-sheet, since the dependency here is more of the pattern of *..abc..abc..abc..*. Both of these types of dependency can be captured by RNRG, in particular, G_1 in Figure 2.15 (a) and G_2 in Figure 6.1, respectively. There are β-sheet structures that contain both of these types of dependency, as shown in Figure 6.3 (c), and the figure actually corresponds to the β-sheet pattern in a picture of a protein structure shown in Figure 6.4. In the figure, the winding line represents the amino acid sequence, and the arrows indicate the β-sheet strands. This actual 3-dimensional picture is drawn based on the 3-dimensional coordinates noted in the PDB (Protein Data Bank) database (Bernstein et al., 1977).

These β-sheet structures can be handled by a grammar like G_1, except each of the trees on the right hand sides of the rewriting rules (α_1) and (α_2) have one of the terminal symbols (the one on the lower right corner) missing. These structures can be combined to obtain larger β-sheets, as is shown in Figure 6.3 (d), and can result in a high degree of complexity. If we use an RNRG of a higher rank, however, such dependencies can be handled. For example, the structure of Figure 6.3 (d) can be expressed by the RNRG(3) grammar given in Figure 6.5.

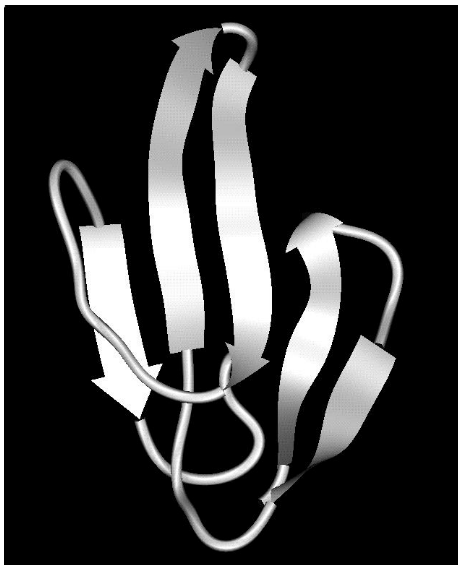


Figure 6.4: 3-dimensional view of an actual β-sheet structure

Note also that there can be insertions of largely irrelevant (i.e. non-β-sheet) regions between the sub-regions resulting in a sequence like $..abcxyyzxabczzxz..$, where x, y, z are irrelevant letters. The irrelevant letters x can be introduced by simple rewriting rules of the form $S \rightarrow s(x, S(\sharp))$ and $S \rightarrow s(S(\sharp), x)$, where $\sharp$ denotes an empty node.

6.3 Learning and Parsing of Restricted Subclass

We here define a linear subclass of RNRG, i.e. linear ranked node rewriting grammar (linear RNRG), as follows:

Definition 6.6 (Linear Ranked Node Rewriting Grammar) (Abe & Mamitsuka, 1997) A *linear ranked node rewriting grammar* G is a 6-tuple $\langle \Sigma_N, \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G \rangle$, where $\langle \Sigma_N \cup \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G \rangle$ is a ranked node rewriting grammar satisfying the following:
(i) Σ_N is a ranked alphabet called the non-terminal alphabet of G.
(ii)Σ_{N_l} is a ranked alphabet such that each symbol in Σ_{N_l} has rank 0 and disjoint from Σ_N. The alphabet Σ_{N_l} is called the *lexical non-terminal alphabet* of G.
(iii) Σ_T is a ranked alphabet such that each symbol in Σ_T has rank 0 and is disjoint from $\Sigma_N \cup \Sigma_{N_l}$. The alphabet Σ_T is called the *terminal alphabet* of G.
(iv) A rewriting rule of G is either
(a) a pair $\langle S_1, \alpha_1 \rangle$ such that $S_1 \in \Sigma_{N_l}$ and $\alpha_1 \in \Sigma_T$ with $rank(\alpha_1) = 0$, or
(b) a pair $\langle S_2, \alpha_2 \rangle$ such that $S_2 \in \Sigma_N$, $rank(S_2) = rank(\alpha_2)$ and the number of non-terminal and terminal symbols in α_2 is at most one and zero, respectively. A rewriting rule $\langle S_1, \alpha_1 \rangle$ is called a *lexical rule*. A rewriting rule $\langle S_2, \alpha_2 \rangle$ is called a *non-lexical rule*. For a non-lexical rule $r_2 = \langle S_2, \alpha_2 \rangle$, in α_2, any partial tree consisting of only $\Sigma_{N_l} \cup \{\lambda, t\}$ has at most one node labeled with t, and the node has two (left and right) outgoing edges to the two nodes labeled with lexical non-terminal or λ. We define the *rank* of G by $rank(G) = \max \{rank(S) \mid S \in \Sigma_N\}$. $\square$

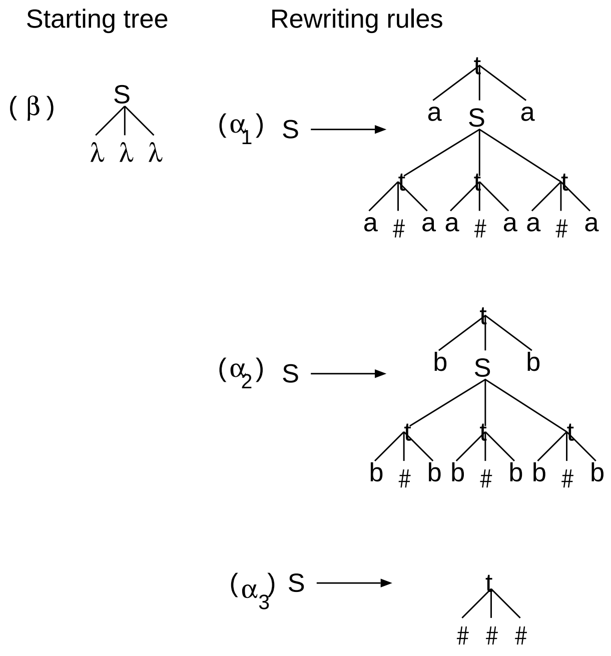


Figure 6.5: An RNRG(3) grammar generating $\{ww^Rww^Rww^Rww^R | w \in \{a, b\}\}$

For a rank 1 linear RNRG, in particular, Definition 6.6 means that there is at most one lexical non-terminal in each of the four corners, i.e. upper left, lower left, lower right and upper right, of the unique (if any) non-terminal on the right hand side. Examples of rank 1 linear RNRG can be found, for example, in Figure 6.8(a). Note that each occurrence of a lexical non-terminal can be thought of as defining a distribution over the terminal alphabet, and this is written in as part of the rewriting rule in the figure. Here, we can define a linear stochastic ranked node rewriting grammar (linear SRNRG) as follows:

Definition 6.7 (Linear Stochastic Ranked Node Rewriting Grammar) (Abe & Mamitsuka, 1997) A *linear stochastic ranked node rewriting grammar G* is a 7-tuple $\langle \Sigma_N, \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G, T_G \rangle$, where:
(i) The 6-tuple $\langle \Sigma_N, \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G \rangle$ is a linear RNRG.
(ii) For a rewriting rule $r = \langle S_r, \alpha_r \rangle$, $T_G(r)$ is a rule application probability of r, where $\sum_{\{r|S_r=S\}} T_G(r) = 1$ is assumed for each $S \in \Sigma_N \cup \Sigma_{N_l}$. We call a rule application probability of a lexical rule a *symbol generation probability.* □

In our current scenario, as we regard the terminal symbols in a linear SRNRG as twenty types of amino acids, we call a symbol generation probability an *amino acid generation probability* and we write a sequence or an amino acid sequence for a terminal symbol sequence. Some structural prediction methods based on SCFG view the amino acid generation probabilities of all lexical non-terminals in a given rewriting rule as part of a joint probability distribution, making it possible to capture the correlations that may exist between them (Sakakibara et al., 1994; Eddy & Durbin, 1994). Although we treat the distribution at each non-terminal as being independent, a comparable effect can be achieved by training multiple copies of the same rewriting rule, as will be reported in Section 6.4.

With the linear constraints described in Definition 6.6, the parsing and learning algorithms for SRNRGs can be significantly simplified. For a general SRNRG of rank k, the entries in the table used in the extended Inside-Outside algorithm (indexed by a $2(k+1)$-dimensional array) need to be node-addresses of the tree structures appearing in the rewriting rules of the grammar, paired with their probabilities of generating the designated sections of the input string. With the linear SRNRG in Definition 6.6, it suffices to place just the non-terminals (and the rewriting rules also in the case of parsing) in the table entries, paired with their probabilities. These constraints also imply that the 'yield' of any particular rewriting rule consists of at most $2(k+1)$ sections in the input string, and hence can be found in 'one shot.' Note that in a general SRNRG, there is no *a priori* limit on how many different sections the yield of any given rewriting rule body can be divided into (though it is bounded above by a constant depending on the grammar) and thus they cannot be found in 'one shot' even if all the $2(k+1)$ tuples of substrings of the input string are inspected.

6.3.1 Learning Algorithm for Linear SRNRG

Our novel learning algorithm is an extension of the Inside-Outside algorithm for SCFG, which is itself an extension of the Baum-Welch algorithm (or the Forward-Backward algorithm) for the Hidden Markov Models. All of these algorithms are local optimization algorithms for the maximum likelihood settings of the rule application probabilities in the input model or grammar. It is well-known (see Levinson, Rabiner, & Sondhi, 1983) that this Baum-Welch re-estimation algorithm is guaranteed to increase the likelihood given to the input sequences, and hence iterative applications of this re-estimation result in a local optimization algorithm.

For both SCFG and linear SRNRG, the forward probability and the backward probability are generalized as inside probability and outside probability. The algorithm we will describe is for rank 1 linear SRNRG. We define the inside probability and outside probability as follows:

Definition 6.8 (Inside Probability) (Abe & Mamitsuka, 1997) Let $G = \langle \Sigma_N, \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G, T_G \rangle$ be a rank 1 linear SRNRG. For a terminal symbol sequence $\sigma = \sigma_1 \cdots \sigma_n$, a non-terminal symbol $S \in \Sigma_N$ and $0 \leq i \leq j \leq k \leq l \leq n$, the *inside probability* $In_\sigma[S, i, j, k, l]$ is the probability that the partially derived tree whose two discontinuous yields match $\sigma_{i+1} \cdots \sigma_j$ and $\sigma_{k+1} \cdots \sigma_l$, contains S as the unique non-terminal. □

Definition 6.9 (Outside Probability) (Abe & Mamitsuka, 1997) Let $G = \langle \Sigma_N, \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G, T_G \rangle$ be a rank 1 linear SRNRG. For a terminal symbol sequence $\sigma = \sigma_1 \cdots \sigma_n$, a non-terminal symbol $S \in \Sigma_N$ and $0 \leq i \leq j \leq k \leq l \leq n$, the *outside probability* $Out_\sigma[S, i, j, k, l]$ is the probability that the partially derived tree whose three discontinuous yields match $\sigma_1 \cdots \sigma_i$, $\sigma_{j+1} \cdots \sigma_k$ and $\sigma_{l+1} \cdots \sigma_n$, contains S as the unique non-terminal. □

Now we show how the inside and outside probabilities are calculated. The inside probabilities at arbitrary index (i, j, k, l) can be defined recursively solely in terms of the inside probabilities of 'smaller' intervals, so they can be calculated as long as the inside probabilities at those (i', j', k', l') such that $(j' - i') + (l' - k') < (j - i) + (l - k)$ have already been calculated. The looping used in the procedure for calculating the inside probabilities exhibited below ensures that this condition is satisfied.

Now let $G = \langle \Sigma_N, \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G, T_G \rangle$ be a given rank 1 linear SRNRG. For a non-lexical rule $r = \langle L(r), \alpha_r \rangle$, let $R(r)$ be the unique non-terminal on the right hand side of r, n^r_f ($f = 1, \cdots, 4$) be the number (0 or 1) of lexical non-terminal symbols at each of the four corners (upper left, lower left, lower right, and upper right) in the α_r, S^r_f be the (if any) lexical non-terminal symbols at a corner

$f = 1, 2, 3, 4$, $P_f^r(a)$ be the symbol generation probability of a lexical rule $S_f^r \rightarrow a$ in a corner f. Note that for a non-lexical rule r and a lexical rule $r' = \langle S_f^r, a\rangle$, $T_G(r') = P_f^r(a)$.

Algorithm 6.10 (Inside) (Abe & Mamitsuka, 1997)
input: rank 1 linear SRNRG $G = \langle \Sigma_N, \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G, T_G\rangle$ and a terminal symbol sequence σ of length n
output: inside probabilities

```
For $i := n$ to 0
  For $j := i$ to $n$
    For $k := n$ to $j$
      For $l := k$ to $n$
        For $S \in \Sigma_N$
```

$$\begin{aligned} In_\sigma[S, i, j, k, l] \quad = \quad & \sum_{\{r \in R_G | L(r)=S\}} \{\ T_G(r) \cdot In_\sigma[R(r), i{+}n_1^r, j{-}n_2^r, k{+}n_3^r, l{-}n_4^r] \cdot \\ & (P_1^r(\sigma_{i+1}))^{n_1^r} (P_2^r(\sigma_j))^{n_2^r} (P_3^r(\sigma_{k+1}))^{n_3^r} (P_4^r(\sigma_l))^{n_4^r}\ \} \end{aligned}$$

□

The outside probabilities can be calculated in a similar fashion, as shown below.

Algorithm 6.11 (Outside) (Abe & Mamitsuka, 1997)
input: rank 1 linear SRNRG $G = \langle \Sigma_N, \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G, T_G\rangle$ and a terminal symbol sequence σ of length n
output: outside probabilities

```
For $i := 0$ to $n$
  For $j := n$ to $i$
    For $k := j$ to $n$
      For $l := n$ to $k$
        For $S \in \Sigma_N$
```

$$\begin{aligned} Out_\sigma[S, i, j, k, l] \quad = \quad & \sum_{\{r \in R_G | R(r)=S\}} \{\ T_G(r) \cdot Out_\sigma[L(r), i{-}n_1^r, j{+}n_2^r, k{-}n_3^r, l{+}n_4^r] \cdot \\ & (P_1^r(\sigma_i))^{n_1^r} (P_2^r(\sigma_{j+1}))^{n_2^r} (P_3^r(\sigma_k))^{n_3^r} (P_4^r(\sigma_{l+1}))^{n_4^r}\ \} \end{aligned}$$

□

Definition 6.12 (Pr) (Abe & Mamitsuka, 1997) Let $G = \langle \Sigma_N, \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G, T_G\rangle$ be a rank 1 linear SRNRG. For a sequence $\sigma = \sigma_1 \cdots \sigma_n$, a non-lexical rule r and $0 \leq i \leq j \leq k \leq l \leq n$, $Pr_\sigma[r, i, j, k, l]$ is the probability that σ is generated by G, using r at the four positions (i, j, k, l) of σ. □

Using the inside and outside probabilities, the $Pr_\sigma[r, i, j, k, l]$ can be calculated as follows:

$$\begin{aligned} Pr_\sigma[r, i, j, k, l] = & Out_\sigma[L(r), i, j, k, l] \cdot In_\sigma[R(r), i{+}n_1^r, j{-}n_2^r, k{+}n_3^r, l{-}n_4^r] \cdot T_G(r) \cdot \\ & (P_1^r(\sigma_{i+1}))^{n_1^r} (P_2^r(\sigma_j))^{n_2^r} (P_3^r(\sigma_{k+1}))^{n_3^r} (P_4^r(\sigma_l))^{n_4^r} \end{aligned}$$

Definition 6.13 (U, V and $\mathcal{V}$) (Abe & Mamitsuka, 1997) Let $G = \langle \Sigma_N, \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G, T_G \rangle$ be a rank 1 linear SRNRG. For a sequence $\sigma = \sigma_1 \cdots \sigma_n$ and a non-lexical rule r, the *weighted average frequency* $U_\sigma(r)$ is the likelihood that σ is generated by G, using rule r. For a sequence $\sigma = \sigma_1 \cdots \sigma_n$, a non-lexical rule r and each corner $f = 1, 2, 3, 4$ of rule r, the *weighted average frequency of terminal symbol* $a \in \Sigma_T$, $V_\sigma^{r,f}(a)$, is the likelihood that σ is generated by G using rule r and that a terminal symbol a is generated at a corner f of rule r. For a sequence $\sigma = \sigma_1 \cdots \sigma_n$, the *weighted average frequency of lexical rule* r', $\mathcal{V}_\sigma(r')$ is the likelihood that σ is generated by G, using rule r'. □

Let $P(\sigma)$ be the likelihood that σ is generated by G. Using $Pr_\sigma[r, i, j, k, l]$ and $P(\sigma)$, the $U_\sigma(r)$ is calculated as follows:

$$U_\sigma(r) \quad = \quad \frac{\sum_i \sum_j \sum_k \sum_l Pr_\sigma[r, i, j, k, l]}{P(\sigma)}$$

Similarly, $V_\sigma^{r,f}(\alpha)$ is calculated as follows (We show the case $f = 1$):

$$V_\sigma^{r,1}(a) \quad = \quad \frac{\sum_i \sum_j \sum_k \sum_l \sum_{\sigma_{i+1}=a} Pr_\sigma[r, i, j, k, l]}{P(\sigma)}$$

For a non-lexical rule r and a lexical rule $r' = \langle S_f^r, a \rangle$, $\mathcal{V}_\sigma(r')$ is calculated as follows:

$$\mathcal{V}_\sigma(r') \quad = \quad \sum_f V_\sigma^{r,f}(a)$$

Proposition 6.14 (Time Complexity of Inside, Outside, U and V)
The time complexity of calculating the inside and outside probabilities for a linear SRNRG is $O(N^4 \cdot M^2)$, where N is the length of a given sequence and M is the number of non-terminal symbols. The time complexity of calculating U and V is also $O(N^4 \cdot M^2)$.
(Proof)
The time complexity of the Inside-Outside algorithm is estimated by the five loops and the summation in Algorithm 6.10 and Algorithm 6.11. The time complexity of calculating U and V also depends on them. □

Finally, the update value for a rule application probability can be re-estimated as follows. Let Ξ be a set of given terminal symbol sequences.

Algorithm 6.15 (Inside-Outside for Linear Stochastic RNRG) (Abe & Mamitsuka, 1997)
input: a set of sequences Ξ and initial rule application probabilities
output: trained rule application probabilities
Repeat the following step until a stopping condition is satisfied, usually until the changes in the probabilities become smaller than a certain preset amount.
1: For a non-lexical rule $r = \langle L(r), \alpha_r \rangle$ and a lexical rule $r' = \langle L(r'), \alpha'_r \rangle$, re-estimate the $T_G(r)$ and $T_G(r')$ using Eqs. (6.1) and (6.2).

$$T_G(r) = \frac{\sum_{\sigma \in \Xi} U_\sigma(r)}{\sum_{\sigma \in \Xi} \sum_{\{q \in R_G | L(q) = L(r)\}} U_\sigma(q)} \tag{6.1}$$

$$T_G(r') = \frac{\sum_{\sigma \in \Xi} \mathcal{V}_\sigma(r')}{\sum_{\sigma \in \Xi} \sum_{\{q' \in R_G | L(q') = L(r')\}} \mathcal{V}_\sigma(q')} \tag{6.2}$$

□

Proposition 6.16 (Time Complexity of Inside-Outside Algorithm)
The time complexity of one iteration of our learning algorithms is $O(N^4 \cdot M^2 \cdot W)$, where W is the number of training sequences, N is the length of a given sequence and M is the number of non-terminal symbols.
(Proof)
As shown in Proposition 6.14, the time complexity of calculating our Inside-Outside algorithm is at most $O(N^4 \cdot M^2)$. Thus, the time complexity of calculating our algorithm for W training sequences is $O(N^4 \cdot M^2 \cdot W)$. □

6.3.2 Reducing Alphabet Size with MDL Approximation

Since there are twenty amino acids and hence the alphabet size is twenty, it is difficult to estimate the symbol generation probabilities at each position with reasonable accuracy with the small data size we have available in practice. Taking advantage of the fact that 'similar' amino acids tend to be easily substitutable, we cluster the amino acids to effectively reduce the alphabet size. The obvious trade-off that we must resolve is between having a fine clustering and thereby gaining high discriminability, and having a coarse clustering and thereby achieving more accurate estimation. In order to resolve this trade-off, we make use of the MDL (Minimum Description Length) principle (Rissanen, 1989), which gives criterion for an optimal clustering, for the given data size.

We now describe our clustering method in some detail. After each iteration of the learning algorithm at each lexical rule, we attempt to merge some of the amino acids, if the merge reduces the total description length which is approximated by the probability parameters calculated up to that point. For this purpose we make use of the Euclidean distance between the 20 amino acids in the (normalized) 2-dimensional space defined by their molecular weight and hydrophobicity. At each iteration, we select the two among the clusters from the previous iteration, which are closest to each other in the above Euclidean space, and merge them to obtain a single new cluster, provided that the merge results in reducing the following approximation of 'description length.'

Definition 6.17 (Description Length of Clusters) (Abe & Mamitsuka, 1997)
Let $G = \langle \Sigma_N, \Sigma_{N_l}, \Sigma_T, \sharp, \beta_G, R_G, T_G \rangle$ be a rank 1 linear SRNRG. We let $\{c^{(j)}\}_{j=1,\cdots,m}$ be a partition of Σ_T, referred to as *clusters*. For a lexical non-terminal S, let $P(c^{(j)}) = \sum_{r|r=\langle S,a\rangle, a\in c^{(j)}} T_G(r)$ be the sum total of terminal symbol generation probabilities in a cluster $c^{(j)}$, where $\sum_{1\leq j\leq m} P(c^{(j)}) = 1$. Let Ξ be a set of sequences. For a lexical rule r, let $L(r)$ be a lexical non-terminal symbol in the left hand side of rule r. For a lexical rule r and a sequence $\sigma \in \Xi$, let $\mathcal{V}_\sigma(r)$ be the weighted average frequency of r. For a lexical non-terminal S and a set of sequences Ξ, we define the effective sample size ε_S as $\varepsilon_S = \sum_{\sigma\in\Xi} \sum_{r'|L(r')=S} \mathcal{V}_\sigma(r')$. For clusters $\{c^{(j)}\}_{j=1,\cdots,m}$ at a lexical rule whose lexical non-terminal is S, we define the approximate description length as follows:

$$-\varepsilon_S \sum_{1\leq j\leq m} P(c^{(j)}) \log \frac{P(c^{(j)})}{m} + \frac{|\Sigma_T| \log \varepsilon_S}{2}. \tag{6.3}$$

□

Note that the above approximation of description length by the average minus logarithmic likelihood of the current values of the rule application probabilities is accurate only if those probability values are reliable. The algorithm keeps merging more clusters in this fashion, but once it fails to merge one pair, it will not try to merge any other pair in the same iteration, in order to ensure that the merge process does not take place too fast.

Figure 6.6(a) exhibiting how the 20 amino acids are distributed in the 2-dimensional space defined by their molecular weights and hydrophobicity, corresponds to the domain $\mathcal{X}_B$ in Chapter 3, which is shown in Figure 3.5. Figure 6.6(b) shows an example of clusters over $\mathcal{X}_B$ obtained by our clustering method in our experiment. We can summarize the algorithm as follows:

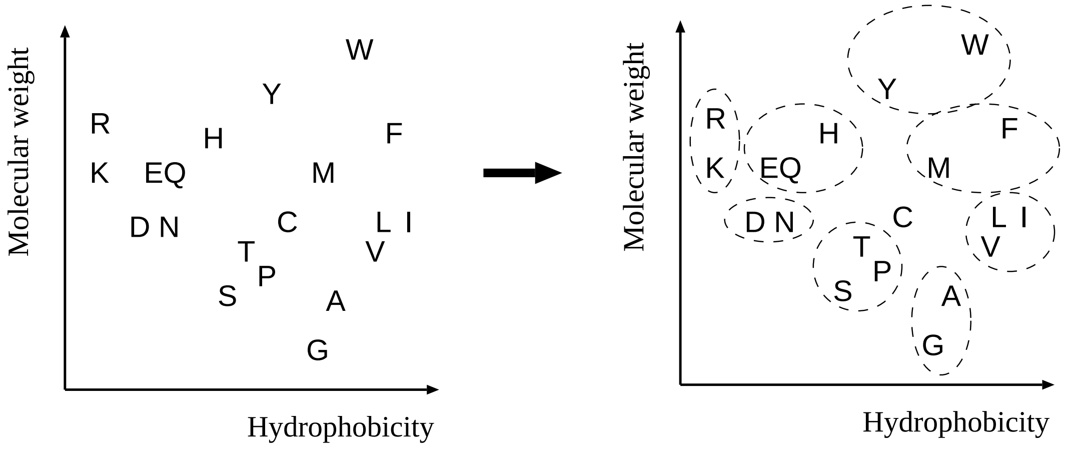


Figure 6.6: (a) Distribution of 20 amino acids in the 2-dimensional space, and (b) an example of a clustering obtained by our method.

Algorithm 6.18 (Reducing Alphabet Size)
input: pair distances of terminal symbols, rule application probabilities
output: clusters of terminal symbols
Perform the following steps at each iteration of the Inside-Outside algorithm.
1: Choose the closest pair among unmerged pairs.
2: Calculate the description length of the clusters when the chosen pair is merged, according to Eq. (6.3).
3: If the merging reduces the description length, merge the pair and go to step 1; otherwise, the merging is suspended and go to the next iteration of the Inside-Outside algorithm. □

6.3.3 Parallel Parsing Algorithm and Prediction

For predicting the β-sheet regions of a test amino acid sequence whose secondary structure is unknown, we use the stochastic tree grammar that has been trained by the learning algorithm on a training data set, and parse the input sequence. We predict the regions generated by the *β-sheet rules*, which are the rewriting rules that correspond to β-sheet, in the most likely parse of the input string to be β-sheet regions.

Definition 6.19 (Parse by Linear SRNRG) (Abe & Mamitsuka, 1997) Let G be a rank 1 linear SRNRG. Let $P(\mathcal{S}|G)$ be the probability that a sequence $\mathcal{S}$ is generated by G. The *most likely parse* is given as follows:

$$arg \max_{\mathcal{S}} P(\mathcal{S}|G).$$

We call $\max_{\mathcal{S}} P(\mathcal{S}|G)$ the *maximum application probability.* □

The parsing algorithm can be generally easily obtained by replacing '$\sum$' by 'max' in the outside algorithm (Algorithm 6.11), and retaining the most likely sub-parse at any intermediate step.

As we noted in Section 6.1, we parallelized our parsing algorithm to run on a 32-processor CM-5. In parallelizing this algorithm, we isolated the data dependency by introducing as the outmost loop parameter, d ($d = (j - i) + (l - k)$ in the rank 1 case), which stands for the total length of all the substrings that are outside those designated by the current indices. That is, we replace the first four **For** loops in the algorithm for calculating the outside probabilities with those shown below.

```
**For** $d := n$ to 0
 **For** $i := 0$ to $n - d$
  **For** $j := i + d$ to $i$
   **For** $l := n$ to $i + d$
```

This way, the computation of all table entries for a given d could in principle be performed in parallel.

Here, in order to reduce the memory requirement of our parsing algorithm, we further placed the following constraints on the grammars.

1. The grammars are designed so that the applications (i.e. derivation) of rewriting rules always take place in the following order.

$$\text{Non-}\beta\text{-sheet rules} \Rightarrow \beta\text{-sheet rules} \Rightarrow \text{Non-}\beta\text{-sheet rules}$$

2. The rule application probabilities of all the non-β-sheet rules are set identically, and the symbol generation probabilities in these rewriting rules are fixed.

With the above constraints placed on the grammars, it is no longer necessary to store the partial probabilities for the applications of non-β-sheet rules, since their contribution to the likelihood of the input sequence depends only on the length of the non-β-sheet regions in the final analysis, since their rule application probabilities are all identical.

This enables us not only to save a great deal of run-time memory, but also to significantly cut down on the computation time as it is no longer necessary to communicate these partial probability values between the parallel processors. Specifically, with the above constraints the parsing algorithm can simply search for those four positions (i, j, k, l) which assign the maximum application probability to the β-sheet rules, given that their rewriting begins at those positions.

At the cost of sacrificing the generality of the parsing algorithm, this makes it possible to handle a good part of the sequences existing in actual databases of length up to 200 or more in realistic computation time.

Let T be the maximum number of processors. For a given test sequence of length n, within the four loops given above, we allocate to each processor t the computation for all $i \in I(t)$, where $I(t)$ is a block of consecutive values of i of size $\lfloor n/T \rfloor$ or $\lceil n/T \rceil$. Now, for any given test sequence σ, and for each processor t, let $L_\sigma[t]$ be the maximum application probability for β-sheet rules calculated at the t-th processor. The algorithm computes in parallel $L_\sigma[t]$ for each processor t, which is to be the maximum of all $L_\sigma[i, j, k, l]$, which are the application probabilities for β-sheet rules at i, j, k, l for all $i \in I(t)$. The details of the algorithm are now exhibited below:

Algorithm 6.20 (Parallel Parsing Algorithm for Linear SRNRG)
input: a sequence σ of length n and a rank 1 linear SRNRG G
output: $Pos(t_{max})$, which are the four positions that give the maximum application probability for β-sheet rules

```
**For** $d := n$ to 0
  **Parallel For** $t := 1$ to $T$
  **For** $i \in I(t)$ (in ascending order)
    **For** $j := i + d$ to $i$
      **For** $l := n$ to $i + d$
        $k := j - i + l - d$.
        Calculate $L_\sigma[i, j, k, l]$.
        **if** $L_\sigma[i, j, k, l] > L_\sigma[t]$ **then**
          $L_\sigma[t] := L_\sigma[i, j, k, l]$ and $Pos(t) := (i, j, k, l)$
$t_{max} := \arg\max_{1 \le t \le T} L_\sigma[t]$
Return $Pos(t_{max})$
```

□

In a given test sequence σ, the output $Pos(t_{max})$ is to be regarded as the *starting positions* of four β-strands which comprise the β-sheet structure pattern expressed by the grammar.

Proposition 6.21 (Time Complexity of Our Parallel Parsing Algorithm)
The time complexity of our parsing algorithm is $O(N^3)$, where N is the length of a given test sequence.
(Proof)
First, as shown by the inside two loops of Algorithm 6.20, at each d and t (i.e. t-th processor), the time complexity of our parsing algorithm is $O(N^2)$. Then, as shown by the most outside loops of Algorithm 6.20, the d takes N to 0 and thus, the total time complexity of our parallel parsing algorithm is $O(N^2 \cdot N)$. □

In Figure 6.7, we show the processing time required by our parallel parsing algorithm, when run on a typical β-sheet grammar fragment on a 32-processor CM-5. Figure 6.7(a) plots the required processing time versus the input sequence length for the sequential and parallel algorithms, and Figure 6.7(b) plots the ratio between them. The speed-up achieved by our parsing algorithm, as

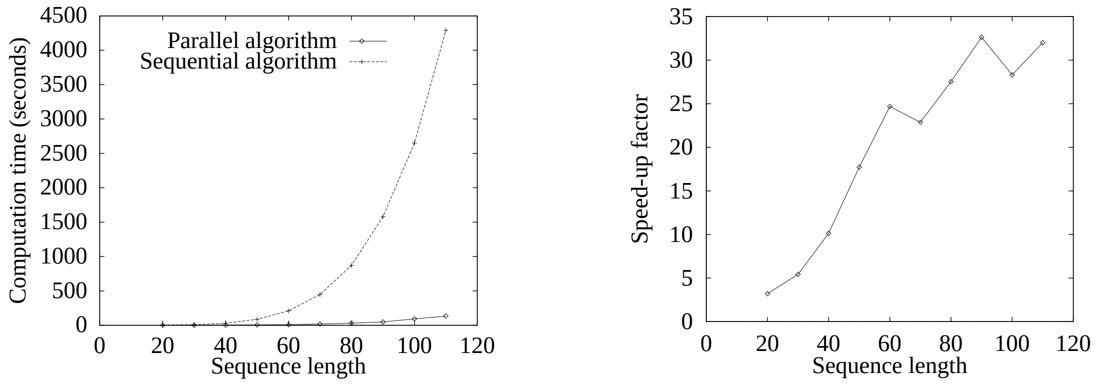


Figure 6.7: The processing time of our parsing algorithm on 32-processor CM-5.

compared to the sequential (simplified) version, is almost linear, and thus a proportionate increase in efficiency would result if the number of processors were increased.

6.4 Experimental Results

6.4.1 Cross-prediction with Structurally Similar Proteins

We applied our method on real data obtained from the HSSP database. In all of our experiments, we used sequences listed in PDB_SELECT 25% list (Hoboem et al., 1992) for both training and test data. PDB_SELECT 25% list is a database containing sequences of protein with known structure possessing at most 25% sequence similarity with one another, and thus it is ensured that no test sequence has more than 25% sequence similarity with any training data. We then enhanced each training sequence with the set of aligned sequences listed in the HSSP(Homology-derived Secondary Structure of Proteins) (Sander & Schneider, 1991) for it.

In our first experiment, we picked three different proteins, 'Fasciculin' (or '1fas' in the code used in PDB_SELECT 25%), 'Caldiotoxin' (1cdt_A) and 'Neurotoxin B' (1nxb), all of which are toxins. These three proteins do have relatively similar structures and their common structure was shown in Figure 6.4. However, their sequences have less than 25% sequence similarity to one another, and hence alignment alone can hardly detect this similarity. We trained a stochastic RNRG with training data consisting effectively of bracketed sequences for one of the three proteins, say 1fas, and used the acquired grammar to predict the location of β-sheet regions in an amino acid sequence of another one of the three, either 1cdt_A or 1nxb. By bracketing the input sequences, we mean that we isolated out the discontinuous substrings of the training sequences that correspond to β-sheets from the rest, and trained the probability parameters of the β-sheet rules in the grammar with them. Bracketed input samples are often used in applications of SCFG in speech recognition. The probability parameters of the non-β-sheet rules were set to be uniform. We then used the acquired stochastic RNRG grammar to parse an amino acid sequence of either 1cdt_A or 1nxb, and predicted the location of β-sheet regions according to where the β-sheet rules are in the most likely parse. It was able to predict the location of all three β-strands contained in the test sequence almost exactly, missing only one or two residues which were absent in all of the training data in both cases. We repeated the same experiment for all six possible combinations of the training data and a test sequence from the three proteins. Our method was able to predict all three of the β-strands in all cases, except in predicting the location of β-sheet in a test sequence for 1cdt_A from training data for 1nxb: It failed to identify one of the three β-strands correctly in this case.

Figure 6.8(a) shows the part of the stochastic RNRG(1) grammar obtained by our learning algorithm on the training set for 1fas that generates the β-sheet regions. Note that, in the figure, the amino acid generation probabilities at each position are written in a box. For example, the distribution at the upper right corner in (α_4) gives probability 0.80 to the cluster [I, L, V] and probability 0.10 to the single amino acid Y. The interpretation of the grammar is summarized schematically in Figure 6.8(b). It is easy to see that the grammar represents a class of β-sheets of type (c) in Figure 6.3. Each of the rewriting rules (α_1), (α_2), (α_3), (α_4), (α_6) and (α_7) generates part of the β-sheet region corresponding to a row of H-bonds, and (α_5) inserts an 'extra' amino acid that does not take part in any H-bond. Rewriting rule (α_4) says that in the third (from the top) row of H-bonds, amino acids I, L and V are equally likely to occur in the leftmost strand, and it is very likely to be K in the middle strand. Note that I, L, and V have similar physico-chemical properties, and it is reasonable that they were merged to form a cluster.

Figure 6.9(a) shows the most likely parse obtained by the grammar on a test sequence of 1cdt_A. The shaded areas indicate the actual β-sheet regions, which are all correctly predicted. The seven types of thick lines correspond to the parts of the most likely parse generated by the seven rewriting rules shown in Figure 6.8(a), respectively. The structural interpretation of this parse is indicated schematically in Figure 6.9(b), which is also exactly correct. Note that the distributions of amino

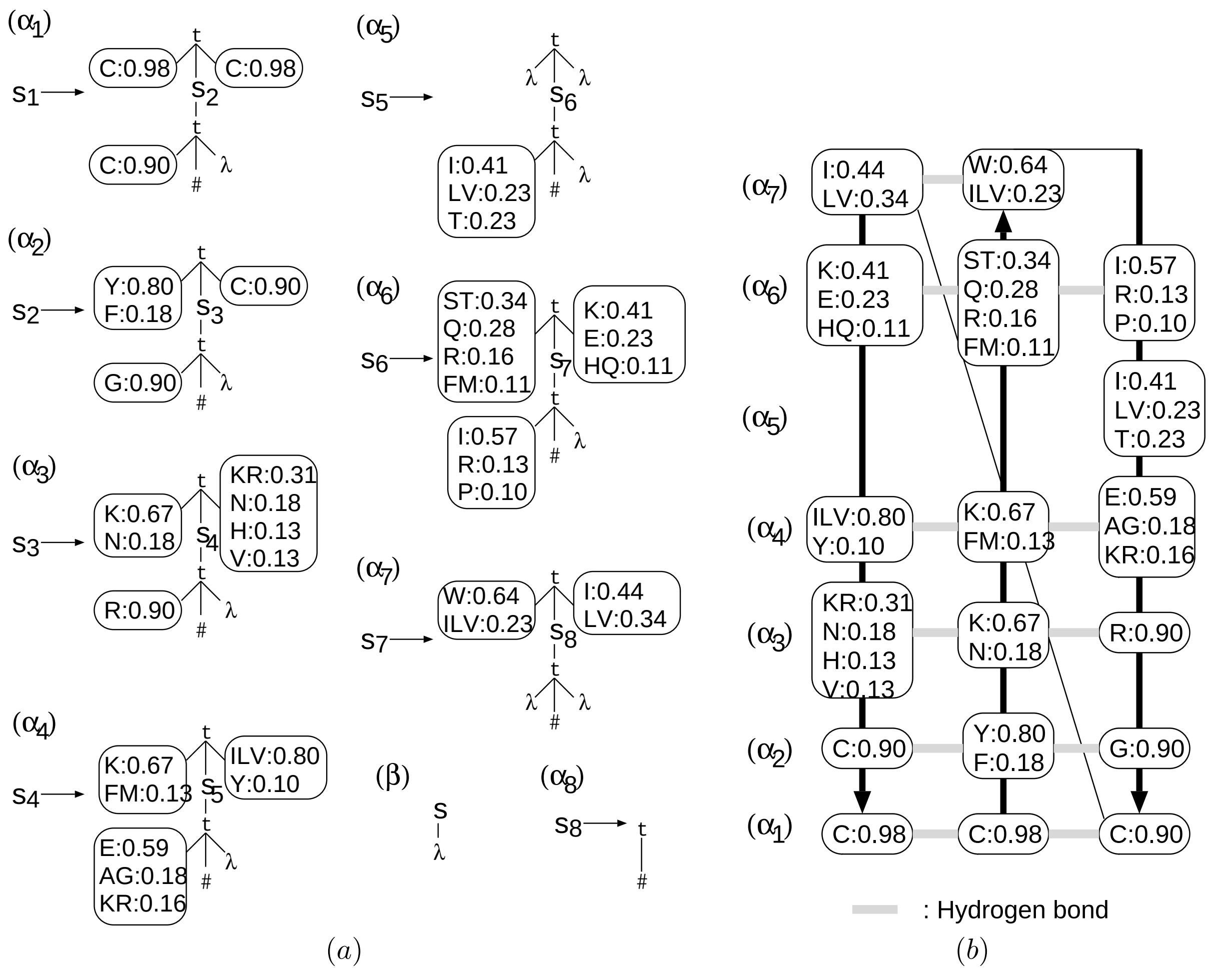


Figure 6.8: (a) A part of the acquired RNRG grammar and (b) its interpretation.

acids are quite well spread over a large number of amino acids. For example, none of the amino acids in the third strand of the test sequence, except the last two Cs, receives a dominantly high probability in the acquired grammar. The merging of I, L and V mentioned above, therefore, was crucial for the grammar to be able to predict the third strand of the β-sheet in the test sequence.

6.4.2 Capturing Correlations with Multiple Rewriting Rules

One apparent shortcoming of the experimental result we just described is that only one copy of each of the rewriting rules $(\alpha_1), \cdots, (\alpha_7)$ was present in the trained grammar. As a result, each of the acquired rewriting rules was able to simply capture the distributions of amino acids at each residue position, and therefore was not able to truly capture the correlations that exist between residue positions, even if they are captured by a single rewriting rule. In another experiment we conducted using exactly the same data as in the above experiment, we used multiple copies (two in particular) of each of the β-sheet rules $(\alpha_1), \cdots, (\alpha_7)$. Note that we used randomly generated numbers for the initial values of their probability parameters. In the acquired grammar, some rewriting rules were split into a pair of rewriting rules that significantly differ from each other, while others became basically two copies of

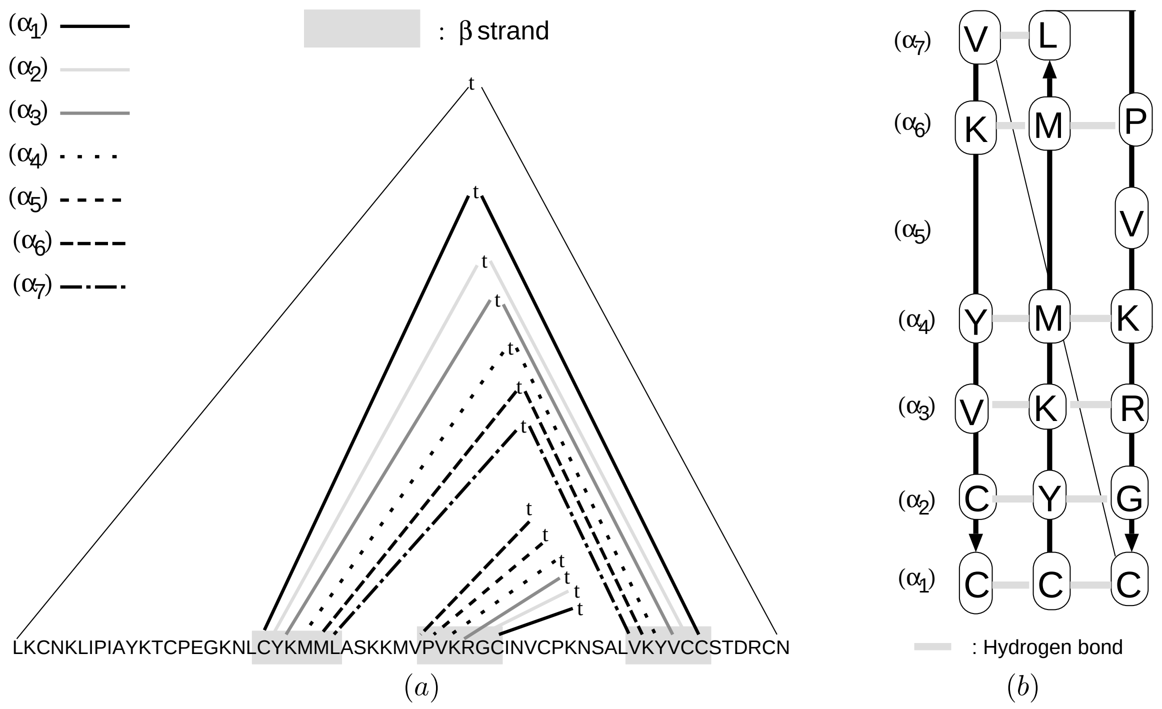
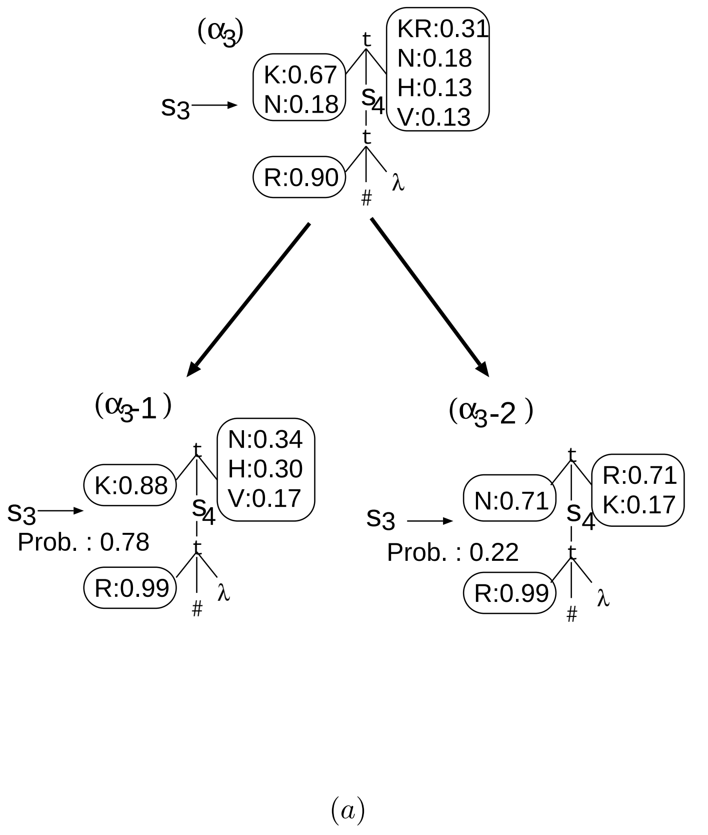


Figure 6.9: (a) The parse of the test sequence and (b) its interpretation.

the same rewriting rule. An example of a rewriting rule that was split is (α_3) in Figure 6.8(a), and the two rewriting rules split into are shown in Figure 6.10(a). This split is meaningful, because in the new grammar, the joint distribution over the two nodes at the top are seen to be heavily concentrated on (K, [N, H]) and (N, [R, K]), which is finer than what we had in the previous grammar ([K, N], [N, H, R, K]). This way, the grammar was able to capture the correlation between these residue positions, which are far from each other in the input sequence.

The grammar containing two copies each of the β-sheet rules obtained using training data for 1fas was used to predict a test sequence for both 1cdt_A and 1nxb. As before, the locations of all three β-strands were predicted exactly correctly. Interestingly, distinct copies of some of the split rewriting rules were used in the respective most likely parses for 1cdt_A and 1nxb. For example, rewriting rule (α_3-1) was used in the most likely parse for the test sequence for 1cdt_A, (α_3-2) for 1nxb. It seems to indicate that the training sequences for 1fas contained at least two dependency patterns for this bonding cite, as shown in Figure 6.10(b), and the corresponding bonding cite in 1cdt_A was of the first type and 1nxb of the second.

The point just illustrated is worth emphasizing. If one tried to capture this type of correlations that exist in bonding cites by a hidden Markov model (HMM), it would necessarily result in a much higher complexity. For example, suppose that eight bonding cites in a row (say each with just two residue positions for simplicity) are split into two distinct rewriting rules. Note that in an HMM, the eight rewriting rules would have to be realized by two copies of consecutive states - sixteen states in a chain. Since there are $2^8 = 256$ possible combinations of rewriting rules to use, the HMM would have to have 256 non-deterministic branches of state sequences, each corresponding to a possible combination of the eight options. In the case of stochastic tree grammar, we only needed to have 2×8 rewriting rules. Clearly this huge saving in complexity is made possible by the richer expressive



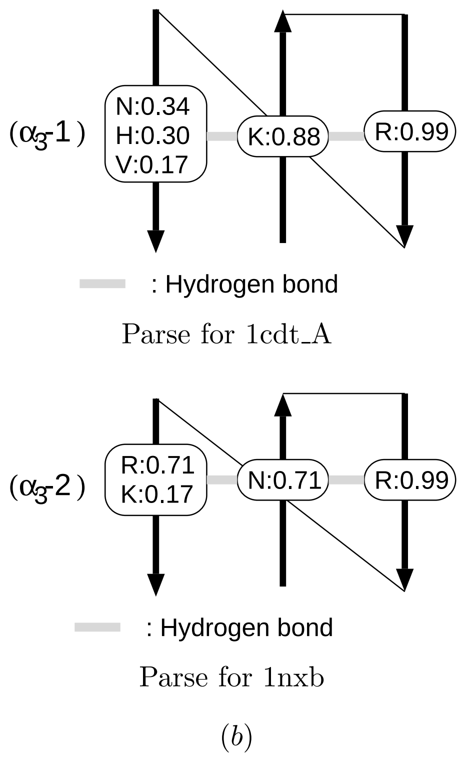


Figure 6.10: (a) Rules (α_3) split into and (b) their interpretations.

power of stochastic tree grammars.

6.4.3 Towards Scientific Discovery

We conducted a large-scale experiment in which using a trained rank 1 SRNRG, we try to search a sequence having the same β-sheet structure pattern as the one used in the trained grammar. The rank 1 four-strand β-sheet patterns we consider here correspond exactly to the class of β-sheet patterns classified as 'two sequentially adjacent hairpin β-strand motifs' in Branden and Tooze (1991), and there are twelve patterns belonging to this class, as shown in Figure 6.11. It is reported in Branden and Tooze (1991) that the frequencies of these twelve patterns found in today's organisms vary greatly. In fact, in the data that we used in our experiment, we only found eight patterns, (a),(b),(c),(d),(e),(g),(j) and (l). Among the training data we used, only six (a),(b),(c),(g),(j) and (l) were found. (c.f. Table 6.1.)

Data Generation

Training Data For training data, we used the aligned data available in HSSP Ver 1.0 (Sander & Schneider, 1991) database of EMBL. HSSP provides alignment data with more than 30% sequence similarity to protein sequences with known structure. In particular, we used all (and therefore unbiased) four-strand patterns satisfying the following conditions as our training data.

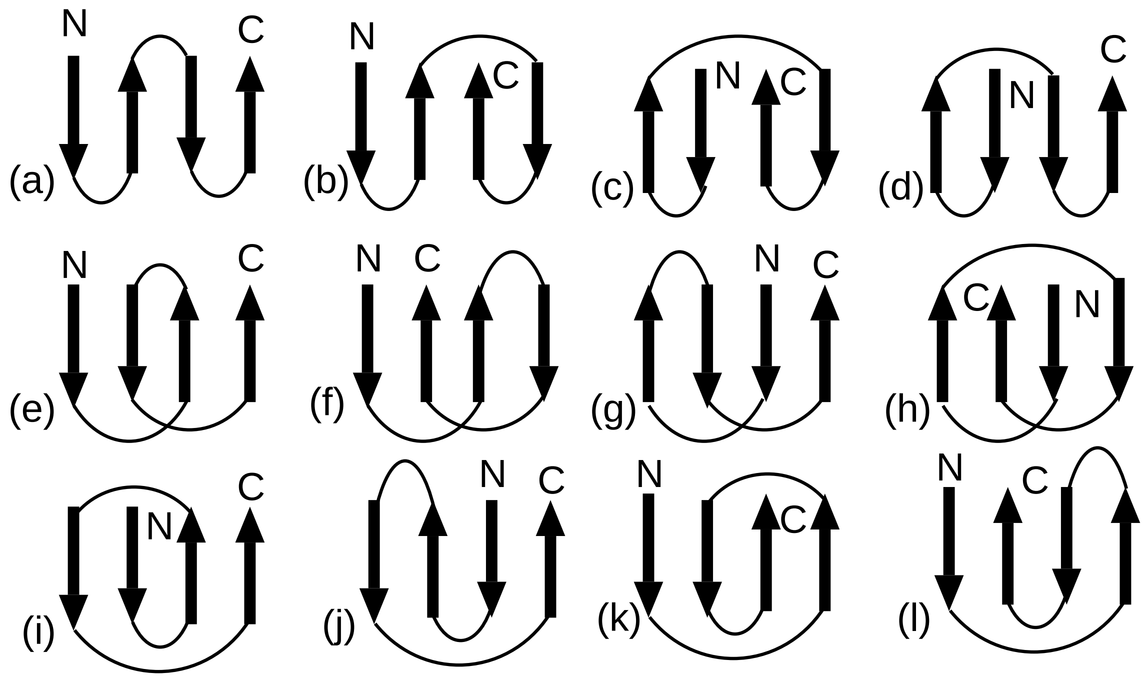


Figure 6.11: Twelve types of rank 1 four-strand patterns

1. The *key sequence,* to which the other sequences in the group are aligned to, is that of a protein contained in the PDB_SELECT25% list (Hoboem et al., 1992).

2. In at least 20 of the sequences aligned to the key sequence as having no more than 70% sequence similarity in HSSP, β-strands corresponding to all four β-strands in the key sequence are present.

3. Each β-strand has length at least 3.

36 four-strand patterns were obtained this way, contained in the 18 proteins listed in Table 6.1. Note that when there are more than one four-strand patterns in a protein, each of these patterns was considered independent and used separately as training data, even when they overlap one another. We call the set of aligned data for each of these patterns a *training data group.*

We show the training data obtained for each four-strand pattern in Table 6.1.

Test Data The test data were obtained by extracting all (and therefore unbiased) sequences satisfying the following conditions.

1. It is contained in the PDB_SELECT25% list (Hoboem et al., 1992).

2. Its length does not exceed 200. Note that this condition is placed for efficiency consideration.

3. It has at least one of the four-strand patterns appearing in the training data.

The 25 test data obtained this way are shown in Table 6.2.

PDB code	Protein	Patterns
1aak	Ubiquitin conjugating enzyme	(a)
1apm_E	C-AMP-dependent protein kinase	(a)
1caj	Carbonic anhydrase II mutant	(a), (b), (j)
1cob_A	Superoxide dismutase	(a), (j), (l)
1dlh_B	HLA-DR1 human class II histocompatibility protein	(a)
1fkb	FK506 binding protein	(c)
1gd1_O	Holo-D-gluceraldehyde-3-phosphate dehydrogenase	(g)
1hdx_A	Alcohol dehydrogenase	(a), (j)
1hle_A	Horse leukocyte elastase inhibitor	(j)
1mct_A	Trypsin	(a), (l)
1mfb_H	Fab fragment	(j)
1ppn	Papain	(a)
1smr_A	Renin	(b), (g)
2cpl	Cyclophilin A	(j)
4blm_A	β-lactamase	(c), (l)
5nn9	Neuraminidase	(a), (l)
6fab_L	Antigen-binding fragment of murine anti-phenylarsonate	(j)
8cat_A	Catalase	(a), (c), (j), (l)

Table 6.1: Training data used in our experiment

Training and Prediction Phase

We manually constructed a tree grammar fragment consisting only of β-sheet rules for each of the 36 training patterns, and then trained the probability parameters in them, using as training data all the aligned sequences for that pattern. In doing so, we set the initial symbol generation probability to be uniform. As we stated earlier, we employed the bracketing technique, namely of extracting the β-strand portions in the training data and trained β-sheet rules each of the above grammar fragment with them.

One point worth mentioning in training is that we tested the effect of having multiple copies of each rewriting rule in the hope that the correlation between residue positions contained in a single rewriting rule can be captured by a mixture of distributions. It was rarely found, however, that the trained parameters of multiple copies of the same rewriting rule resulted in significantly different symbol generation probability distributions. In fact, the data size we had available was barely enough to train a single symbol generation probability distribution at a particular position, and not enough to learn the joint distribution over two or more positions. In our final large-scale experiment, therefore, we only had one copy for each rewriting rule.

The prediction was done by analyzing the input sequence using the grammar fragments each trained on a training data group, and taking the location and structural pattern of the most likely analysis among them all. In the sequel, we refer to this prediction method as 'MAX.'

We partition the proteins appearing in the test data into the following two categories.

1) test_iso : proteins with an isolated four-strand pattern.
   Proteins having a four or five-strand pattern in isolation, and the first and/or the last four of them belong to the class of rank 1 four-strand patterns.

2) test_part : proteins with four-strand patterns existing as part of a larger β-sheet pattern.
   Proteins having one or more rank 1 four-strand patterns and contain at least six β-strands.

Out of the 25 test data group, there were 10 in the former category, and 15 in the latter.

PDB code	Protein	Patterns included in the protein
1bbp_A	Bilin binding protein	(a), (c), (j), (l)
1bet	β-nerve growth factor	(a)
1bfg	Basic fibroblast growth factor mutant	(a)
1bsa_A	Barnase mutant	(a)
1cau_A	Canavalin	(a), (c)
1cau_B	Canavalin	(a), (c)
1cdh	CD4 type I	(a)
1csk_A	C-SRC kinase (SH3 domain)	(a)
1dsb_A	Disulfide bond formation protein	(l)
1epa_B	Epididymal retinoic acid-binding protein	(a), (c), (j), (l)
1gpr	Glucose permease	(a), (j), (l)
1hbq	Retinol binding protein	(a), (c), (j), (l)
1ifc	Intestinal fatty acid binding protein	(a), (c), (j), (l)
1len_A	Lectin	(a)
1lts_D	Heat-labile enterotoxin	(b)
1mdc	Fatty acid binding protein	(a), (c), (l)
1mup	Major urinary protein	(a), (l)
1pts_A	Streptavidin	(a), (c), (j), (l)
1rnd	Ribonuclease A	(a), (c), (j), (l)
1tbp_A	TATA-Binding protein	(l)
1tfi	Transcriptional elongation factor SII	(a)
1tnr_A	Tumor necrosis factor receptor P55	(c)
2snv	Sindbis virus capsid protein	(a), (l)
8i1b	Interleukin 1-β	(a), (j)
9rnt	Ribonuclease T1	(a)

Table 6.2: Test data used in our experiment

Experimental Results

The results of the experiments on the test data in test_iso are shown in Table 6.3-(a). In the table, '#place' denotes the number of strand positions predicted by MAX having non-empty intersections with the actual β-strands of the test sequence, and '#contact' denotes the number of strands among these, which are correctly paired with a sterically neighbouring strand, including their relative orientation. Note that #contact is a very severe criterion, since it focuses on only neighbouring strands and in calculating it, we do not count the correct relations of distant strands in predicted β-sheets.

These results indicate that from #place, 65% of the 40 ($= 4 \times 10$) strand locations that were predicted, had a non-empty intersection with an actual strand, and from #contact, surprisingly roughly one half overall had a correct local structure. This figure compares well against the state-of-the-art protein secondary structure prediction methods, although the size of the test data in our experiment was admittedly small. For example, the accuracy of Riis and Krogh's (1996) method for the three-state prediction problem (distinguishing between α-helix, β-sheet, and others) was about 72 percent. The problem of identifying the β-sheet regions, however, is known to be more difficult (due in part to the long-distance interactions that govern these regions), and, for example, the above method of Riis and Krogh's identified only 57 percent of the β-sheet regions.

Among the 10 test sequences, there were two for which all strands were predicted approximately correctly; 1csk_A and 1tfi. Both of these were predicted by the grammar fragment trained on 1aak, which is a very different protein from both 1csk_A and 1tfi. Figure 6.12 shows a three-dimensional view of the actual structure of 1aak. Figure 6.13 shows the actual β-strands in 1csk_A and those predicted by our method, and Figure 6.14 shows the actual β-strands in 1tfi and those predicted by our method. Note, as before, that the winding line represents the amino acid sequence, and the arrows indicate the β-sheet strands. Thus, our experiment has provided evidence to suspect that

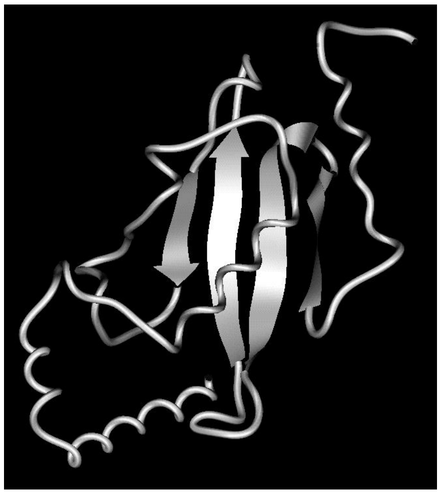


Figure 6.12: 1aak structure (training)

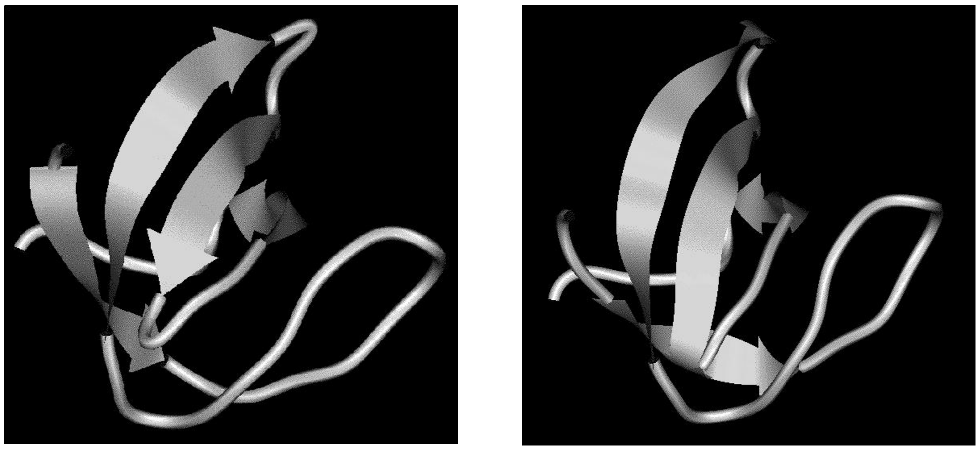


Figure 6.13: (a) Actual 1csk_A structure and (b) Predicted 1csk_A structure.

PDB	MAX	
code	#contact	#place
1bet	2	3
1bfg	2	2
1bsa_A	2	3
1csk_A	4	4
1dsb_A	0	1
1len_A	2	2
1rnd	0	2
1tfi	4	4
1tnr_A	0	2
9rnt	3	3
total	19	26

(a)

PDB	RAND1		RAND2	
code	#contact	#place	#contact	#place
1bet	0.80	2.00	1.10	2.00
1bfg	0.40	1.60	0.40	1.30
1bsa_A	0.70	1.90	0.50	1.20
1csk_A	0.90	2.80	0.70	2.30
1dsb_A	0.20	0.80	0.00	0.80
1len_A	0.60	1.20	0.00	0.50
1rnd	0.80	1.60	0.00	0.80
1tfi	2.10	3.00	1.50	2.70
1tnr_A	0.80	1.60	0.00	1.00
9rnt	0.90	2.10	0.70	1.80
total	8.20	18.60	4.90	14.40

(b)

Table 6.3: Prediction results for the test data in test_iso

the four-strand patterns in these three proteins are perhaps evolutionally related, even though they belong to very different proteins and have no obvious sequence similarity with one another.

In order to assess how significant the predictive performance in the above experiment in terms of #place and #contact is, we conducted a similar experiment using two different random prediction methods.

1) RAND1: It employs the analysis given by a randomly chosen grammar fragment out of the 36 in the above experiment.

2) RAND2: It randomly picks four strand regions, each of length 6 (which is the average length of a β-strand in the training data), and always predicts pattern (a) as the structural pattern, which is the most frequently occurring pattern in the test data.

Note here that basically, MAX should be compared with RAND2, since RAND1 uses the results of our method. The results for these 'random' methods are shown in Table 6.3-(b), each averaged over ten trials. We observe that the total #contact for RAND2 is less than five, and is significantly lower than that of MAX. As for RAND1, its performance is better than RAND2, but it still is no match to that of MAX. These results indicate that the predictive performance of MAX, especially with respect to #contact on test_iso, is statistically significant. We note, by the way, that the test data having #contact exceeding 1 were all 'anti-parallel' β-sheets.

We similarly show the predictive performance of MAX and the two randomized methods on test_part in Table 6.4-(a). Interestingly, in this case, it is observed that no significant difference in predictive performance is observed between all three methods.

These results indicate that the method and the training data used in our experiment can predict the location and structure of relatively isolated four-strand patterns with some significance, but not those that are part of a larger β-sheet structure. It is suspected that more global information would be required for accurate prediction of those larger β-sheet structures.

We also conducted a similar experiment, in which we partitioned the training data into train_iso and train_part as well, and tried to predict the proteins in test_part using only data in train_part. (See Table 6.4-(b).) Surprisingly, #contact was improved (to 16), though #place remained roughly the same. We thus conclude that, at least for prediction of rank 1 four-strand patterns, the prediction of partial patterns is not only more difficult than that of isolated patterns, but for its prediction, the data for the isolated patterns can even degrade the predictive performance. This suggests that it

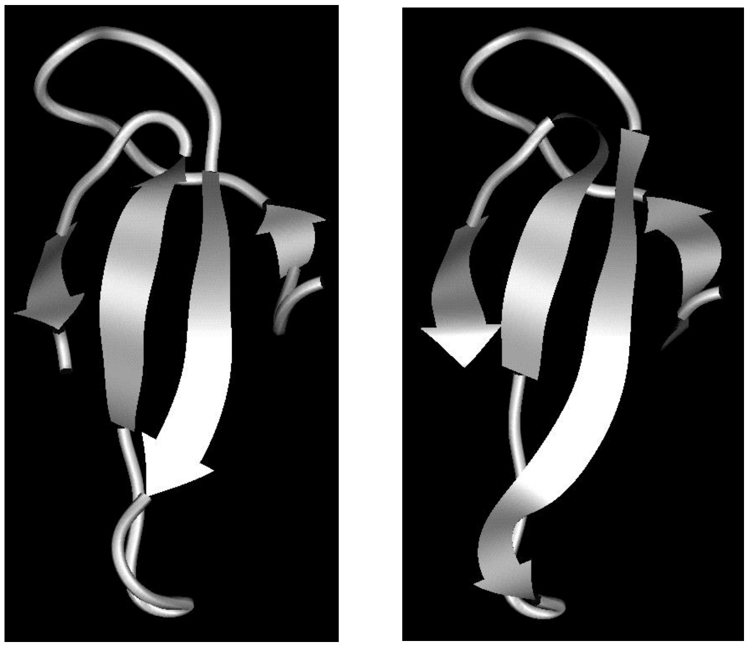


Figure 6.14: (a) Actual 1tfi structure and (b) Predicted 1tfi structure.

PDB	MAX		RAND1		RAND2	
code	#contact	#place	#contact	#places	#contact	#place
1bbp_A	0	2	1.00	2.70	0.60	1.90
1cau_A	0	3	0.20	1.80	0.40	2.50
1cau_B	0	1	0.00	1.40	0.20	1.50
1cdh	0	1	0.60	1.40	0.20	1.20
1epa_B	2	3	1.30	2.80	0.80	1.90
1gpr	0	1	0.40	1.80	0.00	2.00
1hbq	2	3	0.60	2.60	0.70	2.70
1ifc	0	2	1.50	3.80	1.30	3.10
1lts_D	0	1	0.80	2.00	0.40	1.40
1mdc	0	3	1.40	3.20	1.10	2.50
1mup	0	2	0.90	2.00	0.50	2.00
1pts_A	3	4	0.70	3.60	1.10	3.50
1tbp_A	0	3	0.40	2.60	0.20	2.00
2snv	2	3	0.70	2.70	0.70	2.60
8i1b	0	2	0.40	2.20	0.80	1.50
total	9	34	10.90	36.60	9.00	32.30

(a)

PDB	MAX	
code	#contact	#place
1bbp_A	3	3
1cau_A	0	2
1cau_B	0	1
1cdh	2	2
1epa_B	0	3
1gpr	0	1
1hbq	2	2
1ifc	0	3
1lts_D	2	3
1mdc	0	3
1mup	2	3
1pts_A	3	4
1tbp_A	0	3
2snv	2	3
8i1b	0	1
total	16	37

(b)

Table 6.4: Prediction results for the test data in test_part

may be better to have generalized patterns for the entire partial patterns, rather than the four-strand patterns that are part of them. In the tree grammar terms, this would require SRNRGs of a higher rank, and therefore would mean a higher computational burden.

Comparison with the SCOP Protein Classification

As we stated in 6.1, we checked to see how proteins used as training and test data in our experiment are classified according to an existing protein classification. We used the classification in the SCOP database (Murzin et al., 1995), which classifies proteins with respect to their structures.

The SCOP database is organized as a classification tree, and a code consisting of the following fields are used to specify each category in the classification.

Root . Class . Fold . Superfamily . Family . Protein

Of the proteins used in our experiments, all but a couple of exceptions are classified into distinct Superfamily categories. Recall that there were two proteins, 1csk_A and 1tfi, for which all four strands were approximately correctly predicted, including their patterns (See Table 6.3-(a)) and these correct predictions were made by the grammar fragment for 1aak. The SCOP codes for these proteins are as follows.

1aak	1cska	1tfi
1.004.019.001.001.001	1.002.023.002.001.007	1.007.030.002.001.001

These three proteins are not classified in the same category, even at the highest level of 'Class,' and yet they were cross-predicted approximately correctly.

This observation indicates that there are cases when two proteins are of very different protein classes and are classified very far in a standard classification such as SCOP, and yet share relatively similar local structures (β-sheets), which can be exploited for cross-prediction. Thus, with relatively few data that are available at present, it makes more sense to use our prediction method as a tool for scientific discovery, in an attempt to find β-sheet structures that are commonly shared by non-homologous proteins.

6.5 Conclusion

We defined a novel class of stochastic tree grammars and established a new learning strategy for predicting protein secondary structures using the grammars. Our experimental results show that our method can be used to discover common β-sheet structures shared between proteins without sequence similarity. In the absence of notable sequence similarity, it is difficult to predict the β-strand regions in the test sequences by existing secondary structure prediction methods. Even more difficult is to determine their structural patterns, something that is not done by the usual secondary structure prediction framework. In our experiments, the three proteins, of which two are approximately correctly predicted by the third, are classified into three different classes. Hence, such a prediction is not possible by existing prediction methods such as 'homology modeling' or 'remote homology modeling.' Furthermore, they provide positive evidence of the potential of our method as a tool for scientific discovery that allows us to discover unnoticed structural similarity in proteins having little or no sequence similarity.

Chapter 7

Concluding Remarks and Future Perspective

7.1 Summary

We defined four stochastic knowledge representations, i.e. a stochastic rule with finite partitioning, a probabilistic network with finite partitionings, a hidden Markov model and a stochastic ranked node rewriting grammar, and established new learning algorithms for the four knowledge representations. The learning algorithms we established are the following three:
1) methods based on the Minimum Description Length (MDL) criterion,
2) a gradient method for minimizing a type of error-distance function, and
3) a method based on the Expectation-Maximization (EM) algorithm.

The methods based on the MDL criterion were proposed to estimate the number of cells in a stochastic rule with finite partitioning in Chapter 3, to estimate the structure of a probabilistic network with finite partitionings in Chapter 4, and to cluster an alphabet in Chapter 6. These algorithms have a significant characteristic that allows us to estimate not only probability parameters in a given stochastic model but also the stochastic model itself. In other words, a number of features hidden in given training examples can be automatically found by this type of model estimation.

The gradient method for minimizing a type of error-distance function was proposed to estimate probability parameters of hidden Markov models (HMMs) in Chapter 5. This algorithm allows us to perform supervised learning, which has never been done for HMMs, and improve the predictive performance of HMMs. Similar algorithms were applied to train neural networks, which were used to compare the performance of neural network methods with that of our methods, in Chapters 3, 4 and 5.

The learning algorithm based on the EM algorithm was proposed to estimate probability parameters of stochastic ranked node rewriting grammars in Chapter 6. No learning algorithms had ever previously been proposed for the grammars, and the algorithm helps make it possible to efficiently estimate the parameters of such complicated grammars. A similar learning algorithm for HMMs, called the Baum-Welch, was applied in Chapter 5 for purposes of comparison with our supervised learning algorithm for HMMs.

We evaluated these methods by computer experiments, using actual biological sequences. Experimental results showed that all of the knowledge representations defined and the new learning algorithms established greatly contribute to the crucial problems of computational molecular biology. In Chapter 3, for the problem of predicting α-helices, our method achieved a prediction accuracy of nearly the same level as that of the best method at that time. In Chapter 4, from a number of sequences of a particular motif, our method found a number of features hidden in the motif. In Chapter 5, our supervised learning method for HMMs improves the predictive performance of the conventional algorithm for HMMs, and for the problem of predicting peptides binding to an MHC molecule, our method achieves a predictive performance which exceeds those of the existing methods applied to the problem. As described in Chapter 6, our method was actually able to capture the long distance dependency in β-sheet regions in a way that had not been possible using any earlier method.

7.2 Future Directions

We here describe possible future work to improve our four learning strategies.

Predicting α-helices with Stochastic Rule Learning

1) *Correlations among residues.*
   In the SR method, we assumed for simplicity's sake that all residues in an α-helix region of a training protein are probabilistically independent (see Eq. (3.1)). However, it is rather more natural to suppose that there may exist some mutual correlation among the residues which affect their secondary structures. In fact, there are several methods that consider a mutual correlation among residues in an α-helix (c.f. Gibrat, Garnier, & Robson, 1987), and the neural network learning methods themselves are correlation-based methods. Considering residue interactions in the SR method will improve its predictive performance. One possible way for this improvement is the probabilistic network with finite partitionings given in Chapter 4, in which inter-residue relations are automatically obtained by our learning algorithm. This network, however, has not been applied to predicting α-helices. This is an important topic which remains for future study.

2) *Considering for long-range interactions*
   Our experiments in Chapter 3 showed that the average prediction accuracy (81%) and the α-helix content rate for the test proteins only slightly differed from those for the training proteins (84% prediction accuracy). This suggests that for the α-helix prediction methods based on the local properties in the primary structure, we have come close to the limit of prediction accuracy. Considering long-range interactions in the primary structure in addition to its local properties is indispensable to improve the current predictive performance, and is a challenging future issue. We note that the stochastic ranked node rewriting grammar proposed in Chapter 6 is one method for capturing such long-distance dependencies. The challenge is to apply such a grammar, or propose a new method, which can capture such long-distance dependencies to further improve the predictive performance.

Learning Probabilistic Networks

Supervised learning for probabilistic network

In Chapter 4, we train a probabilistic network with finite partitionings from given positive examples. In short, we perform unsupervised learning for the network. Thus, in Chapter 4, the average prediction accuracy of our networks is slightly lower than that of neural networks trained by a supervised learning method. This indicates that if we could perform supervised learning to train our network, a higher prediction accuracy might be obtained. Supervised learning for the probabilistic network with finite partitionings is a possible future project.

Supervised Learning of Hidden Markov Models

Faster learning algorithm

In our supervised learning algorithm, the computation time per iteration is on the same order as that of the Baum-Welch algorithm. However, the number of iterations required in our algorithm until the updated parameter values converge is far larger than that of the Baum-Welch. Thus, a possible future work is to propose a faster algorithm for supervised learning of HMMs.

Learning Stochastic Tree Grammars

1) *Cutting down on computational requirement.*

   The most important future challenge is to reduce the rather high computational requirement of our parsing algorithm, which has prohibited us to date from conducting full scale experiments in which the β-sheet structure of an arbitrary amino acid sequence is predicted.

2) *Stochastic grammar with richer expressive power.*

   As is shown in Chapter 6, a linear stochastic ranked node rewriting grammar is suitable for representing the structures of β-sheets, but if we deal with more complicated residue contacts in a protein, we have to consider more general grammars than a linear stochastic ranked node rewriting grammar. Thus, another possible future work related to stochastic tree grammars is to define a more general stochastic grammar which has a richer expressive power than that of a linear stochastic ranked node rewriting grammar, but for which efficient learning and parsing algorithms can be proposed. We believe that this type of research direction will greatly contribute to the field of computer science as well as computational molecular biology.

Bibliography

Abe, N. (1988). Feasible learnability of formal grammars and the theory of natural language acquision. In *Proceedings of the International Conference on Computational Linguistics*, pp. 1–6 Budapest, Hungary.

Abe, N., & Mamitsuka, H. (1994). A new method for predicting protein secondary structures based on stochastic tree grammars. In *Proceedings of the 11th International Conference on Machine Learning*, pp. 3–11 New Brunswick, NJ. Morgan Kaufmann.

Abe, N., & Mamitsuka, H. (1997). Predicting protein secondary structure using stochastic tree grammars. *Machine Learning*, *29*, 275–301.

Abe, N., & Mamitsuka, H. (1998). Query learning strategies using boosting and bagging. In *Proceedings of the 15th International Conference on Machine Learning*, pp. 1–9 Madison, WI. Morgan Kaufmann.

Altschul, S. F., Boguski, M. S., Gish, W., & Wooton, J. C. (1994). Issues in searching molecular sequence database. *Nature Genetics*, *6*, 119–129.

Anfinsen, C. B. (1973). Principles that govern the folding of protein chains. *Science*, *181*, 223–230.

Asai, K., Hayamizu, S., & Handa, K. (1993). Prediction of protein secondary structure by the hidden Markov model. *Comput. Applic. Biosci.*, *9*, 141–146.

Bahl, L. R., Brown, P. F., de Souza, P. V., & Mercer, R. L. (1986). Maximum mutual information estimation of hidden Markov models parameters for speech recognition. In *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing*, Vol. 1, pp. 49–52.

Bairoch, A., & Apweiler, R. (1996). The SWISS-PROT protein sequence data bank and its new supplement TrEMBL. *Nucl. Acids Res.*, *24*, 21–25.

Bairoch, A., & Apweiler, R. (1999). The SWISS-PROT protein sequence data bank and its supplement TrEMBL in 1999. *Nucl. Acids Res.*, *27*(1), 49–55.

Bairoch, A., P.Bucher, & Hofmann, K. (1996). The PROSITE database, its status in 1996. *Nucl. Acids Res.*, *24*, 189–196.

Baldi, P., & Chauvin, Y. (1994a). Hidden Markov models of the G-protein-coupled receptor family. *J. Comput. Biol.*, *1*(4), 311–326.

Baldi, P., & Chauvin, Y. (1994b). Smooth on-line learning algorithms for hidden Markov models. *Neural Comp.*, *6*, 307–318.

Baldi, P., Chauvin, Y., Hunkapillar, T., & McClure, M. (1994). Hidden Markov models of biological primary sequence information. *Proc. Natl. Acad. Sci.*, *91*, 1059–1063.

Barker, W. C., Garavelli, J. S., McGarvey, P. B., Marzec, C. R., Orcutt, B. C., Srinivasarao, G. Y., Yeh, L-S. L., Ledley, R. S., Mewes, H-W., Pfeiffer, F., Tsugita, A., & Wu, C. (1999). The PIR-international protein sequence database. *Nucl. Acids Res.*, *27*(1), 39–43.

Barron, A. R., & Cover, T. M. (1991). Minimum complexity density estimation. *IEEE Trans. on Information Theory (IT)*, *IT-37*, 1034–1054.

Barton, G. J. (1995). Protein secondary structure prediction. *Curr. Opin. Struct. Biol.*, *5*, 372–376.

Barton, G. J., & Sternberg, M. J. E. (1987). A strategy for rapid multiple alignment of protein sequences. *J. Mol. Biol.*, *198*, 327–337.

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, *3*, 1–8.

Benson, D., Boguski, M., Lipman, D. J., Ostell, J., Ouelette, B. F. F., Rapp, B. A., & Wheeler, D. L. (1999). GenBank. *Nucl. Acids Res.*, *27*(1), 12–17.

Berger, M. P., & Munson, P. J. (1991). A novel randomized iterative strategy for aligning multiple protein sequences. *Comput. Applic. Biosci.*, *7*, 479–484.

Bernstein, F., Koetzle, T., Williams, G., Meyer, E., Brice, M., Rodgers, J., Kennard, O., Shimanouchi, T., & Tasumi, M. (1977). The Protein Data Bank: a computer-based archival file for macromolecular structures. *J. Mol. Biol.*, *112*, 535–542.

Boczko, E. M., & Brooks III, C. L. (1995). First-principles calculation of the folding free energy of a three-helix bundle protein. *Science*, *269*, 393–396.

Bohr, H., Bohr, J., Brunek, S., Cotterill, M., Lautrup, B., Norskov, L., Olsen, H., & Petersen, B. (1988). Protein secondary structure and homology by neural networks. *FEBS Letters*, *241*(1,2), 223–228.

Bork, P., & Koonin, E. V. (1996). Protein sequence motifs. *Curr. Opin. Struct. Biol.*, *6*, 366–376.

Branden, C., & Tooze, J. (1991). *Introduction to Protein Structure.* Garland Publishing, Inc., New York and London.

Bridle, J. S. (1990). Alpha-Nets: A recurrent 'neural' network architecture with a hidden Markov model interpretation. *Speech Comm.*, *9*, 83–92.

Brown, M., Hughey, R., Krogh, A., Mian, I. S., Sjölander, K., & Haussler, D. (1993). Using dirichlet mixture priors to derive hidden Markov models for protein families. In *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology*, pp. 47–55 Menlo Park, CA. AAAI press.

Brünger, A. T., & Nilges, M. (1993). Computational challenges for macromolecular structure determination by X-ray crystallography and solution NMR-spectroscopy. *Quart. Rev. Biophys.*, *26*, 49–125.

Brusic, V., Rudy, G., Kyne, A. P., & Harrison, L. C. (1997). MHCPEP, a database of MHC-binding peptides: update 1996. *Nucl. Acids Res.*, *25*(1), 269–271.

Casari, G., Andrade, M. A., Bork, P., Boyle, J., Daruvar, A., Ouzounis, C., Schneider, R., Tamames, J., Valencia, A., & Sander, C. (1994). Challenging times for bioinformatics. *Nature*, *376*, 647–648.

Chothia, C. (1992). One thousand families for the molecular biologist. *Nature*, *357*, 543–544.

Chou, P. Y., & Fasman, G. D. (1974a). Conformational parameters for amino acids in helical, β-sheet, and random coil regions calculated from proteins. *Biochemistry*, *13*(2), 211–221.

Chou, P., & Fasman, G. (1974b). Prediction of protein conformation. *Biochemistry*, *13*(2), 222–245.

Churchill, G. A. (1989). Stochastic models for heterogeneous DNA sequences. *Bull. Math. Biol.*, *51*, 79–94.

Cohen, F. E., Abarbanel, R. M., Kuntz, I. D., & Fletterrick, R. J. (1986). Turn prediction in proteins using a pattern matching approach. *Biochemistry*, *25*, 266–275.

Cooper, G. F., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*, 309–347.

Cuff, J. A., & Barton, G. J. (1999). Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. *Proteins: Struct. Funct. Genet.*, *34*(4), 508–519.

Dayhoff, M. O., Schwartz, R. M., & Orcutt, B. C. (1978). A model of evolutionary change in protein. *NBRF*, *5*, 363–373.

Delcher, A. L., Kasif, S., Goldberg, H. R., & Hsu, W. H. (1993). Protein secondary structure modelling with probabilistic networks. In *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology*, pp. 109–117 Menlo Park, CA. AAAI press.

Dempster, A. P., Laird, N., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm(with discussion). *Journal of the Royal Statistical Society*, *B-39*, 1–38.

Doolittle, R. F. (1986). *Of URFs and ORFs: A primer on how to analyze derived amino acid sequences.* University Science Books, Mill Valley, CA.

Doolittle, R. F., Feng, D. F., Johnson, M. S., & McClure, M. A. (1986). Relationships of human protein sequences to those of other organisms. *Cold Spring Harbor Symp. Quant. Biol.*, *51*, 447–455.

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

Eddy, S. R. (1996). Hidden Markov models. *Curr. Opin. Struct. Biol.*, *6*, 361–365.

Eddy, S. R., & Durbin, R. (1994). RNA sequence analysis using covariance models. *Nucl. Acids Res.*, *22*, 2079–2088.

Eddy, S. R., Mitchison, G., & Durbin, R. (1995). Maximum discrimination hidden Markov models of sequence consensus. *J. Comput. Biol.*, *2*(1), 9–24.

Falk, K., Rötzschke, O., Stevanović, S., Jung, G., & Rammensee, H.-G. (1991). Allele-specific motifs revealed by sequencing of self-peptides eluted from MHC molecules. *Nature*, *351*, 290–296.

Fauchere, J., & Pliska, V. (1983). Hydrophobic parameters of amino acid side chains from the partitioning of N-acetyl-amino acid amides. *Eur. J. Med. Chem. Chim. Ther.*, *18*, 369–375.

Fletcher, R. (1987). *Practical Methods of Optimization: Second Edition.* John Wiley & Sons, New York.

Frishman, D., & Argos, P. (1997). Seventy-five percent accuracy in protein secondary structure prediction. *Proteins: Struct. Funct. Genet.*, *27*(3), 329–335.

Garnier, J., Osguthorpe, D., & Robson, B. (1978). Analysis of the accuracy and implication of simple methods for predicting the secondary structure of globular proteins. *J. Mol. Biol.*, *120*, 97–120.

Gibrat, J., Garnier, J., & Robson, B. (1987). Further developments of protein secondary structure prediction using information theory. *J. Mol. Biol.*, *198*, 425–443.

Gotch, F., McMichael, A., & Rothbard, J. (1988). Recognition of influenza A matrix protein by HLA-A2-restricted cytotoxic T lymphocytes. Use of analogues to orientate the matrix peptide in the HLA-A2 binding site. *J. Exp. Med.*, *168*, 2045–2057.

Grate, L. (1995). Automatic RNA secondary structure determination with stochastic context-free grammars. In *Proceddings of the third Internation Conference on Intelligent Systems for Molecular Biology*, pp. 136–137 Menlo Park, CA. The AAAI Press.

Gribskov, M., Lüthy, R., & Eisenberg, D. (1990). Profile analysis. *Methods Enzymol.*, *183*, 146–159.

Gribskov, M., McLachlan, A. D., & Eisenberg, D. (1987). Profile analysis: Detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA*, *84*, 4355–4358.

Gulukota, K., Sidney, J., Sette, A., & DeLisi, C. (1997). Two complementary methods for predicting peptides binding major histocompatibility complex molecules. *J. Mol. Biol.*, *267*, 1258–1267.

Hall, P., & Hannan, E. J. (1988). On the stochastic complexity and nonparametric density estimation. *Biometrika*, *75*, 705–714.

Hayward, S., & Collins, J. F. (1992). Limits on α-helix prediction with neural network models. *Proteins: Struct. Funct. Genet.*, *14*, 372–381.

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, *20*, 197–244.

Hoboem, U., Scharf, M., Schneider, R., & Sander, C. (1992). Selection of a representative set of structures from the Brookhaven Protein Data Bank. *Prot. Sci.*, *1*, 409–417.

Hofmann, K., Bucher, P., Falquet, L., & Bairoch, A. (1999). The PROSITE database, its status in 1999. *Nucl. Acids Res.*, *27*(1), 215–219.

Holley, L., & Karplus, M. (1989). Protein secondary structure prediction with a neural network. *Proc. Natl. Acad. Sci.*, *86*, 152–156.

Hubbard, T., Park, J., Lahm, A., Leplae, R., & Tramontano, A. (1996). Protein structure prediction: Playing the fold. *Trends in Biochemical Sciences*, *21*, 279–281.

Hubbard, T. J. P., & Park, J. (1995). Fold recognition and ab initio structure predictions using hidden Markov models and β-strand potentials. *Proteins: Struct. Funct. Genet.*, *23*, 398–402.

Hunter, L. (1993). Molecular biology for computer scientists. In Hunter, L. (Ed.), *Artificial intelligence and molecular biology*, chap. 1. AAAI Press.

Ishikawa, M., Toya, T., Hoshida, M., Nitta, K., Ogiwara, A., & Kanehisa, M. (1993). Multiple sequence alignment by parallel simulated annealing. *Comput. Applic. Biosci.*, *9*, 267–273.

Jelinik, F., Lafferty, & Mercer, R. (1990). Basic methods of probabilistic context free grammars. *IBM Research Reports*, *RC16374(#72684)*.

Jones, D. T. (1997). Progress in protein structure prediction. *Curr. Opin. Struct. Biol.*, *7*, 377–387.

Joshi, A. K., Levy, L., & Takahashi, M. (1975). Tree adjunct grammars. *J. Comp. Sys. Sci.*, *10*, 136–163.

King, R., & Sternberg, M. (1990). Machine learning approach for the prediction of protein secondary structure. *J. Mol. Biol.*, *216*, 441–457.

Klingler, T. M., & Brutlag, D. L. (1994). Discovering side-chain correlation in α-helices. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, pp. 236–243 Menlo Park, CA. AAAI press.

Kneller, D. G., Cohen, F. E., & Langridge, R. (1990). Improvements in protein structure prediction by an enhanced neural network. *J. Mol. Biol.*, *214*, 171–182.

Kraft, L. G. (1949). A device for quantizing, grouping and coding amplitude modulated pulses. Master's thesis, Dept. of Electrical Engineering, M.I.T., Cambridge, MA.

Krogh, A. (1997). Two methods for improving performance of an HMM and their application for gene finding. In *Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology*, pp. 179–186 Halkidiki, Greece. AAAI press.

Krogh, A., Brown, M., Mian, I. S., Sjölander, K., & Haussler, D. (1994a). Hidden Markov models in computational biology. Applications to protein modeling. *J. Mol. Biol.*, *235*, 1501–1531.

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

Krogh, A., & Mitchison, G. (1995). Maximum entopy weighting of aligned sequences of proteins or DNA. In *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, pp. 215–221 Menlo Park, CA. AAAI press.

Krogh, A., & Riis, S. K. (1999). Hidden neural networks. *Neural Comp.*, *11*(2), 541–563.

Langley, P., Provan, G. M., & Smyth, P. (1997). Learning with probabilistic representations. *Machine Learning*, *29*, 91–101.

Lathrop, R. H., Webster, T. A., & Smith, T. F. (1987). ARIADINE:pattern directed inference and hierarchical abstraction in protein structure recognition. *Commun. A. C. M.*, *30*, 909–921.

Levinson, S. E., Rabiner, L. R., & Sondhi, M. M. (1983). An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *The Bell System Technical Journal*, *62*(4).

Lim, V. I. (1974). Algorithms for prediction of α-helices and β-structural regions in globular proteins. *J. Mol. Biol.*, *88*, 873–894.

Lüthy, R., Xenarios, I., & Bucher, P. (1994). Improving the sensitivity of the sequence profile method. *Prot. Sci.*, *3*, 139–146.

Mamitsuka, H. (1993). Representing inter-residue dependencies in protein sequences using probabilistic networks. In *Proceedings of Genome Informatics Workshop IV*, pp. 46–55 Tokyo, Japan. Universal Academy Press.

Mamitsuka, H. (1995). Representing inter-residue dependencies in protein sequences with probabilistic networks. *Comput. Applic. Biosci.*, *11*(4), 413–422.

Mamitsuka, H. (1996). A learning method of hidden Markov models for sequence discrimination. *J. Comput. Biol.*, *3*(3), 361–373.

Mamitsuka, H. (1997). Supervised learning of hidden Markov models for sequence discrimination. In *Proceedings of the First International Conference on Computational Molecular Biology*, pp. 202–208 Santa Fe, NM. ACM press.

Mamitsuka, H. (1998). Predicting peptides that bind to MHC molecules using supervised learning of hidden Markov models. *Proteins: Struct. Funct. Genet.*, *33*(4), 460–474.

Mamitsuka, H., & Abe, N. (1994a). Predicting location and structure of beta-sheet regions using stochastic tree grammars. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, pp. 276–284 Palo Alto, CA. The AAAI press.

Mamitsuka, H., & Abe, N. (1994b). Prediction of beta-sheet structures using stochastic tree grammars. In *Proceedings of Genome Informatics Workshop V*, pp. 19–28 Tokyo, Japan. Universal Academy Press.

Mamitsuka, H., & Yamanishi, K. (1992). Protein secondary structure prediction based on stochastic-rule learning. In *Proceedings of the Third Workshop on Algorithmic Learning Theory*, pp. 240–251 Tokyo, Japan.

Mamitsuka, H., & Yamanishi, K. (1993). Protein α-helix region prediction based on stochastic-rule learning. In *Proceedings of the 26th Hawaii International Conference on System Sciences*, Vol. 1, pp. 659–668 Maui, HI. IEEE Computer Society Press.

Mamitsuka, H., & Yamanishi, K. (1995). α-helix region prediction with stochastic rule learning. *Comput. Applic. Biosci.*, *11*(4), 399–411.

Mattews, B. W. (1975). Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochim. Biophys. Acta.*, *405*, 442–451.

Meinke, D. W., Cherry, J. M., Deans, C., Rounsley, S. D., & Koornneef, M. (1998). *Arabidopsis thaliana*: A model plant for genome analysis. *Science*, *282*, 662–6820.

Michalski, R. S. (1986). Understanding the nature of learning : Issues and research directions. In Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.), *Machine Learning vol.2*. Morgan Kaufmann Publishers, Inc.

Minsky, M. (1986). *The society of mind*. Simon and Schuster, New York.

Moncrief, N. D., Kretsinger, R. H., & Nakayama, S. (1990). Evolution of EF-hand calcium-modulated proteins. I. Relationships based on amino acid sequences. *J. Mol. Evol.*, *30*, 522–562.

Moult, J., Hubbard, T., Bryant, S. H., Fidelis, K., & Pedersen, J. T. (1997). Critical assessment of methods of protein structure prediction (CASP2): Round II. *Proteins: Struct. Funct. Genet.*, *28*(s1), 2–6.

Muggleton, S., King, R. D., & Sternberg, M. J. E. (1992). Protein secondary structure prediction using logic-based machine learning. *Protein Eng.*, *5*, 647–657.

Murzin, A., Brenner, S., Hubbard, T., & Chotia, C. (1995). SCOP: A structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, *247*, 536–540.

Nagano, K. (1977). Triplet information in helix prediction applied to the analysis of super-secondary structures. *J. Mol. Biol.*, *109*, 251–274.

Nakayama, S., Moncrief, N. D., & Kretsinger, R. H. (1992). Evolution of EF-hand calcium-modulated proteins. II. domains of several subfamilies have diverse evolutionary histories. *J. Mol. Evol.*, *34*, 416–448.

Neapolitan, R. (1989). *Probabilistic reasoning in expert systems*. John Wiley & Sons, New York.

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.*, *48*, 443–453.

O'Donoghue, S., & Rost, B. (1995). Computational tools for experimental determination and theoretical prediction of protein structure. Tutorial for ISMB-95.

Olson, M. V. (1995). A time to sequence. *Science*, *270*, 394–396.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems*. Morgan Kaufman Publishers, Inc.

Peitsch, M. C., & Boguski, M. S. (1990). Is Apolipoprotein a mammalian bilin-binding protein?. *New Biol.*, *2*, 197–206.

Peitsch, M. C., & Boguski, M. S. (1991). The first lipocalin with enzymatic activity. *Trends Biochem. Sci.*, *16*, 363–363.

Pennisi, E. (1999). Academic sequencers challenge celera in a sprint to the finish. *Science*, *283*, 1822–1823.

Petersen, S. B., Bohr, H., Bohr, J., Brunek, S., Cotterill, R. M. J., Fredholm, H., & Lautrup, B. (1990). Training neural networks to analyse biological sequences. *Trends in Biotechnol.*, *8*, 304–308.

Presnell, S. R., Cohen, B. I., & Cohen, F. E. (1992). A segment-based approach to protein secondary structure prediction. *Biochemistry*, *31*, 983–993.

Qian, N., & Sejnowski, T. (1988). Predicting the secondary structure of globular proteins using neural network models. *J. Mol. Biol.*, *202*, 865–884.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. of the IEEE*, *77*(2), 257–286.

Riis, S. K., & Krogh, A. . (1996). Improving prediction of protein secondary structure using structured neural networks and multiple sequence alignments. *J. Comput. Biol.*, *3*(1), 163–184.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, *14*, 465–471.

Rissanen, J. (1989). *Stochastic complexity in statistical inquiry*, Vol. 15 of *Comp. Sci.* World Scientific Publishing Company.

Rost, B., & Sander, C. (1993a). Prediction of protein secondary structure at better than 70 % accuracy. *J. Mol. Biol.*, *232*, 584–599.

Rost, B., & Sander, C. (1993b). Secondary structure prediction of all-helical proteins in two states. *Protein Eng.*, *6*, 831–836.

Rost, B., Sander, C., & Schneider, R. (1994). PHD - an automatic mail server for protein secondary structure prediction. *Comput. Applic. Biosci.*, *10*(1), 53–60.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Sakakibara, Y., Brown, M., Hughey, R., Mian, I. S., Sjölander, K., Underwood, R. C., & Haussler, D. (1994). Stochastic context-free grammars for tRNA modeling. *Nucl. Acids Res.*, *22*, 5112–5120.

Sánchez, R., & Sali, A. (1997). Advances in comparative protein-structure modelling. *Curr. Opin. Struct. Biol.*, *7*, 206–214.

Sander, C., & Schneider, R. (1991). Database of homology-derived structures and the structural meaning of sequence alignment. *Proteins: Struct. Funct. Genet.*, *9*, 56–68.

Schabes, Y. (1992). Stochastic lexicalized tree adjoining grammars. In *Proceedings of the International Conference on Computational Linguistics*, pp. 426–432.

Schreiber, F. (1985). The Bayes Laplace statistics of the multinomial distributions. *AEU*, *39*(5), 293–298.

Searls, D. B. (1993). The computational linguistics of biological sequences. In Hunter, L. (Ed.), *Artificial Intelligence and Molecular Biology*, chap. 2. AAAI Press.

Sette, A., Vitiello, A., Farness, P., Furze, J., Sidney, J., Claverle, J. M., Grey, H. M., & Chesnut, R. (1991). Random association between the peptide repertoire of A2.1 class I and several different DR class II molecules. *J. Immunol.*, *147*, 3893–3900.

Shavlik, J. (Ed.)., Proceedings of ML (1998). *Proceedings of the Fifteenth International Conference on Machine Learning.* Morgan Kaufmann Publishers, Inc. ISBN : 1-55860-556-8.

Shortle, D. (1995). Protein fold recognition. *Nature Struct. Biol.*, *2*, 91–93.

Simon, H. A. (1983). Why should machines learn?. In Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.), *Machine Learning : An Artificial Intelligence*. Morgan Kaufmann Publishers, Inc.

Sternberg, M. J. E. (1991). Library of common protein motifs. *Nature*, *349*, 111–111.

Stoesser, G., Tuli, M. A., Lopez, R., & Sterk, P. (1999). The EMBL nucleotide sequence database. *Nucl. Acids Res.*, *27*(1), 18–24.

Stolorz, P., Lapedes, P., & Xia, Y. (1992). Predicting protein secondary structure using neural net and statistical methods. *J. Mol. Biol.*, *225*, 363–377.

Stultz, C. M., White, J. V., & Smith, T. F. (1993). Structural analysis based on state-space modeling. *Prot. Sci.*, *2*, 305–314.

Sugawara, H., Miyazaki, S., Gojobori, T., & Tateno, Y. (1999). DNA data bank of Japan dealing with large-scale data submission. *Nucl. Acids Res.*, *27*(1), 25–28.

Taubes, G. (1996). Software mathcmakers help make sense of sequences. *Science*, *273*, 588–590.

Taylor, W. R. (1986a). The classification of amino acid conservation. *J. Theor. Biol*, *119*, 205–218.

Taylor, W. R. (1986b). Identification of protein sequence homology by consensus template alignment. *J. Mol. Biol*, *188*, 233–258.

Taylor, W. R. (1987). Multiple sequence alignment by a pairwise algorithm. *Comput. Applic. Biosci.*, *3*, 81–87.

The *C.elegans* consortium (1998). Genome sequence of the nematode *C. elegans*: A platform for investigating biology. *Science*, *282*, 2012–2018.

Umemura, Y., Hasegawa, A., Kobayashi, S., & Yokomori, T. (1999). Tree adjoining grammars for RNA structure prediction. *Theoretical Computer Science*, *210*(2), 277–303.

Vijay-Shanker, K., & Joshi, A. K. (1985). Some computational properties of tree adjoining grammars. In *Proceedings of 23rd Meeting of the Association for Computational Linguistics*, pp. 82–93 Chicago, IL.

Voet, D., & Voet, J. G. (1990). *Biochemistry*. John Willey, New York.

White, J. V., Smith, C. M., & Smith, T. F. (1994). Protein classification by stochastic modeling and optimal filtering of amino-acid sequences. *Math. Biosci.*, *119*, 35–75.

Wüthrich (1989). Protein structure determination in solution by nuclear magnetic resonance spectroscopy. *Science*, *243*, 45–50.

Yamanishi, K. (1992). A learning criterion for stochastic rules. *Machine Learning*, *9*, 165–203.

Yamanishi, K., & Konagaya, A. (1991). Learning stochastic motifs from genetic sequences. In *Proceedings of the 8th International Workshop on Machine Learning*, pp. 467–471 Evanston, IL. Morgan Kaufmann.

Zhang, X., Mesirov, J. P., & Waltz, D. L. (1992). Hybrid system for protein secondary structure prediction. *J. Mol. Biol.*, *225*, 1049–1063.