Synthesizing Regularity Exposing Attributes in Large Protein Databases

by

Michael de la Maza
mdlm@ai.mit.edu

B.S., Massachusetts Institute of Technology (1992)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1993

Signature of Author .....
Department of Electrical Engineering and Computer Science
May 7, 1993

Certified by .....
Patrick Henry Winston
Director, MIT Artificial Intelligence Laboratory
Thesis Supervisor

Accepted by .....
Campbell L. Searle
Chairman, Department Committee on Graduate Students

Synthesizing Regularity Exposing Attributes in Large Protein Databases

by

Michael de la Maza

mdlm@ai.mit.edu

Submitted to the Department of Electrical Engineering and Computer Science
on May 7, 1993, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

This thesis describes a system that synthesizes regularity exposing attributes from large protein databases. After processing primary and secondary structure data, this system discovers an amino acid representation that captures what are thought to be the three most important amino acid characteristics (size, charge, and hydrophobicity) for tertiary structure prediction. A neural network trained using this 16 bit representation achieves a performance accuracy on the secondary structure prediction problem that is comparable to the one achieved by a neural network trained using the standard 24 bit amino acid representation. In addition, the thesis describes bounds on secondary structure prediction accuracy, derived using an optimal learning algorithm and the probably approximately correct (PAC) model.

Thesis Supervisor: Patrick Henry Winston
Title: Director, MIT Artificial Intelligence Laboratory

Acknowledgments

Thanks to Patrick Winston, Michael Bolotski, Lawrence Hunter, Michael Jones, Kimberle Koile, Rick Lathrop, Oded Maron, Ken Rice, Bruce Tidor, David Wetherall, Tau-Mu Yi, and Xiru Zhang.

Thanks to Patrick Winston and Rick Lathrop. The author is an NSF Graduate Fellow. This work was supported by generous grants from the Pittsburgh and Illinois National Centers for Supercomputing. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-91-J-4038.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The structure of a protein can be described at four levels: primary structure, secondary structure, tertiary structure, and quaternary structure [Branden and Tooze, 1991]. The primary structure of a protein is the sequential list of amino acids that comprise the protein. Interacting amino acids form units of secondary structure, called alpha helices and beta sheets. Alpha helices and beta sheets are repeating structures that can be identified in the tertiary structure, which is a three-dimensional model of the protein that assigns Cartesian coordinates to each atom in the protein. The relationship among tertiary structure units in a protein is called quaternary structure.

Finding the tertiary structure of a protein is an important step in elucidating its function. Currently, the two methods used for finding tertiary structure, X-ray crystallography and nuclear magnetic resonance are expensive and time-consuming. Although there are thousands of primary structures known, only a few hundred tertiary structures have been determined and only about 50 new ones are determined each year [Lander et al., 1991]. The determination of each structure is still considered a major event.

Predicting secondary structure is thought to be an important step in determining tertiary structure from primary structure. Since 1988, researchers have used machine

learning techniques, primarily neural networks, to predict secondary structure. The best programs have accuracies of 60% to 70%. This thesis continues this line of research. To be precise, the secondary structure prediction problem is:

> Produce an algorithm that given as input the primary sequence of a protein and a distinguished amino acid in that protein produces as output the secondary structure assignment (*alpha helix*, *beta sheet*, or *coil*) of the distinguished amino acid in the protein.

This thesis has two parts. The first part suggests that secondary structure prediction algorithms that use unannotated data require a prohibitive number of instances to achieve a high performance accuracy. The second part describes a database mining system that rediscovers important biochemical properties of amino acids in secondary structure data.

1.2 The problem and a possible solution

Consider a straightforward classification problem. A set of proteins is canvassed for proteins that exhibit a particular function and the question is asked: "Is it possible to separate the functional proteins from the non-functional ones?" Certainly, a sufficiently powerful learning algorithm will be able to take as input the Cartesian coordinates of each of the atoms in a protein and produce a procedure that separates the functional and non-functional proteins. Unfortunately, no such algorithms exist. In addition, this algorithm may produce a procedure that provides no insight into the underlying mechanism.

Consider the following alternative. Instead of asking the learning algorithm to produce a solution in one pass, ask the algorithm to create new synthesized attributes, composed of the input attributes, that have explanatory or predictive power. The search for synthesized attributes may be driven by heuristics[Lenat, 1976, Langley, 1980] or by some task-dependent metric, as it is in this thesis.

More generally, the problem of discovering hidden attributes in data is a fundamental one. As in the above example, objects in databases are not always annotated

with features that facilitate understanding. Only through a process of feature combination can the regularities in the data be made explicit.

1.3 A clear criteria for success

How should a system that claims to synthesize regularity exposing attributes be judged?

We suggest two criteria for success:

- The synthesized attributes should capture interesting properties that can be explained by other means.
- The synthesized attributes should be superior, along some significant dimension, to the original input attributes.

1.4 The result

A system that meets both of these criteria is shown in Figure 1-1.

The goal of this system is to use the primary and secondary structure of proteins to create amino acids representations that facilitate secondary structure prediction. Each amino acid is represented as a bit string. So, a representation for all of the amino acids consists of twenty bit strings.

In the first step, a genetic algorithm searches the space of amino acid representations. The quality of each representation is quantified by training a neural network to predict secondary structure using that representation. The genetic algorithm then uses the performance accuracy of the representation to guide its search and to create amino acid representations that improve the performance accuracy.

In the second step, the best amino acid representation produced by the genetic algorithm during one run is divided into bit strings (one for each amino acid) and these bit strings are clustered using Hamming distance. These clusters capture similarities among the representations for each of the amino acids.

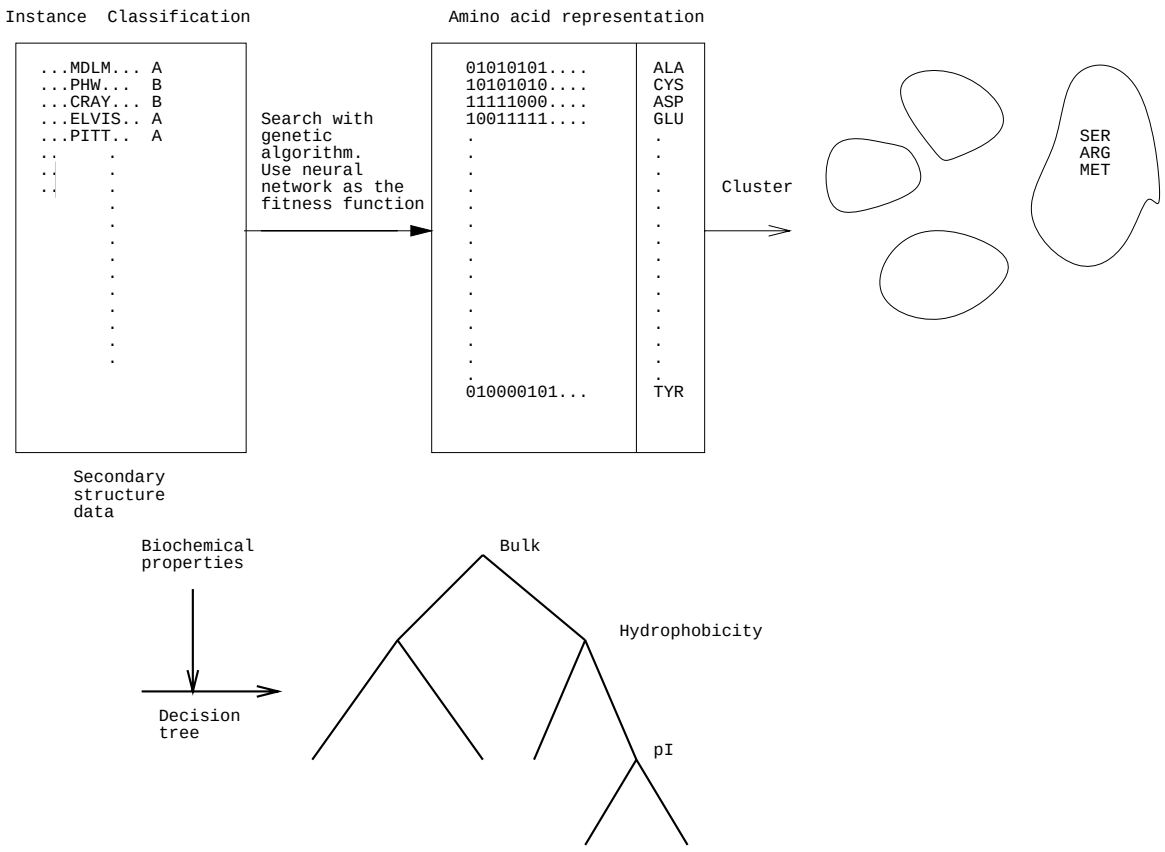

Figure 1-1: A bird's eye view of the system that is fully explained in Chapter 3.

In the third step, a decision tree system uses the clustering and a database of biochemical properties to produce a decision tree that classifies the amino acids using biochemical properties. This decision tree explains why the amino acid representation is a good one. Unlike current work in secondary structure prediction, the end result of this system is not just a technique for predicting secondary structure, but rather a technique that predicts secondary structure and provides an account of why it can do so. Because this system makes explicit what the representation is capturing, it provides more information about protein structure than other algorithms.

The end result of this procedure is an amino representation that, as shown in Chapter 3, represents, in part, bulk, hydrophobicity, and charge (pI) and their interdependencies. This representation achieves a performance accuracy on the secondary structure prediction problem that is comparable to the one achieved by the standard amino acid representation, thus meeting both of the criteria discussed in the previous

section.

1.5 Related work

The first part of this thesis applies results from computational learning theory to the secondary structure prediction problem and the second part combines four learning algorithms (a neural network, a genetic algorithm, a clustering algorithm, and a decision tree procedure) into a database mining system that synthesizes regularity exposing attributes in secondary structure data. This system is used to find regularities that facilitate secondary structure prediction. This related work section considers each of these three topics in turn.

1.5.1 Computational learning theory

The primary theoretical tool for the analysis of learning from examples is the probably approximately correct (PAC) model [Valiant, 1984]. One of the primary features of the PAC model is that it permits analysis of hypotheses that only approximate the correct solution. The PAC model has been extended in many fruitful ways (e.g., [Amsterdam, 1988, Schapire, 1991]) and specific results are available for neural networks [Haussler, 1989], which are the most widely used classification algorithms in secondary structure prediction.

1.5.2 Database mining

Genetic algorithms

In this thesis a genetic algorithm [Holland, 1975, Goldberg, 1989] is used to search the space of possible amino acid representations. As discussed in Chapter 3, other global optimization algorithms may be used instead.

Given the choice of genetic algorithms, there are two questions that are of particular importance. First, how good are the solutions produced by genetic algorithms? Second, how long does it take to produce the solutions?

There are few theoretical results that support genetic algorithms, although the situation is improving (see, e.g., [Thomas and Principe, 1991, de la Maza and Tidor, 1993]), so the justification for using genetic algorithms comes from a twenty year history of producing good empirical results.

Genetic algorithms have produced better than best known traveling salesman solutions [Grefenstette et al., 1985, Whitley et al., 1989], outperformed standard nonlinear programming algorithms [Michalewicz, 1992], and improved searches for criminal suspects [Caldwell and Johnston, 1991]. Of course, genetic algorithms do not always find better solutions than other algorithms (see, e.g., [Quinlan, 1988]).

Neural networks

Neural networks have played a dominant role in secondary structure prediction research since 1988 and, therefore, to facilitate comparisons, they are used in this work.

Theoretical results in neural networks are mixed. Large neural networks have been shown to be Turing equivalent [Sun et al., 1991, Jones, 1992], but training a simple threshold neural network is NP-complete [Blum and Rivest, 1992]. Fast algorithms are known for finding good neural network topologies [Roy and Mukhopadhyay, 1992], but the number of instances needed to train them is typically large [Baum and Haussler, 1989].

Clustering algorithm

Clustering algorithms group objects in such a way that intragroup similarities are maximized while intergroup similarities are minimized. These groups partition the set of all objects so that previously unseen objects may be placed into a group. Thus, the result of running a clustering algorithm on a set of data is not just a grouping of the objects initially available to the program but also a function that maps objects to groups.

A wide range of clustering algorithms have been described and analyzed. AutoClass, a Bayesian clustering algorithm [Cheeseman et al., 1988], assigns to each object a probability that it is in a particular group, unlike most clustering algorithms

which make fixed assignments. COBWEB [Fisher, 1987], the clustering algorithm used in this thesis, is an incremental clustering algorithm that produces hierarchical clusterings. Several other algorithms are statistical in nature. Michalski and Stepp [Michalski and Stepp, 1992] review clustering algorithms.

Decision tree system

Decision tree systems generate trees which, in their simplest form, have single attribute tests on their nodes and classes on their leaves. As with clustering algorithms, there are many decision tree systems, of which the best known are CART [Breiman et al., 1984] and C4.5 [Quinlan, 1993].

C4.5, the decision tree system used in this thesis, has been developed over an extended period of time and includes techniques for reducing the effect of noise and generating production rules from trees.

1.5.3 Secondary structure prediction

Chou and Fasman [Chou and Fasman, 1974] and Lim [Lim, 1974] proposed the secondary structure prediction problem almost twenty years ago. Recently, researchers have used artificial intelligence techniques to attack the problem [Qian and Sejnowski, 1988, Holley and Karplus, 1989].

Zhang *et al.*[Zhang et al., 1992] describe a system that uses a neural network, called Combiner, to combine the predictions of three experts, a neural network, a memory based reasoning system, and a Bayesian statistical module. Each of the three experts examines a thirteen residue "window" in the protein sequence and predicts the secondary structure of the middle residue. The predictions of these three experts are then fed into the Combiner which produces the final prediction of the algorithm. Individually, the neural net had a performance accuracy of 63.1%, the memory based reasoning system had a performance accuracy of 64.5%, and the statistical module had a performance accuracy of 63.5%. The Combiner increased the performance accuracy to 66.4%.

1.6 What this thesis is not about

This thesis does not describe a secondary structure prediction algorithm that has a higher performance accuracy than other algorithms, nor does it claim to do so. Although the database mining system discussed in this thesis was motivated by the secondary structure prediction problem, it has not been fine-tuned to achieve high performance on this task and, as such, the system may be applicable to other domains.

Furthermore, we do not claim that the synthesized attributes created by the system described in this thesis are in any way optimal nor do we claim that they will be useful in other domains in which amino acid representations are important.

1.7 Summary of thesis

This section summarizes the key ideas and main results in this thesis. Chapter 2 addresses the question "How many instances does a secondary structure prediction algorithm need to predict with high accuracy?" and Chapter 3 describes a database mining system that rediscovers important amino acid properties.

1.7.1 How many instances does a secondary structure prediction algorithm need?

Theoretical results from machine learning can be applied to the secondary structure prediction problem to discover how many instances need to be processed in order to achieve a certain performance accuracy. Why is this helpful? These theoretical results can:

- Highlight shortcomings in current approaches that otherwise would not be uncovered.
- Suggest fruitful avenues for new investigation.

In the first part of chapter 2, the question of how many instances are required to achieve a certain performance accuracy is first explored by creating an optimal

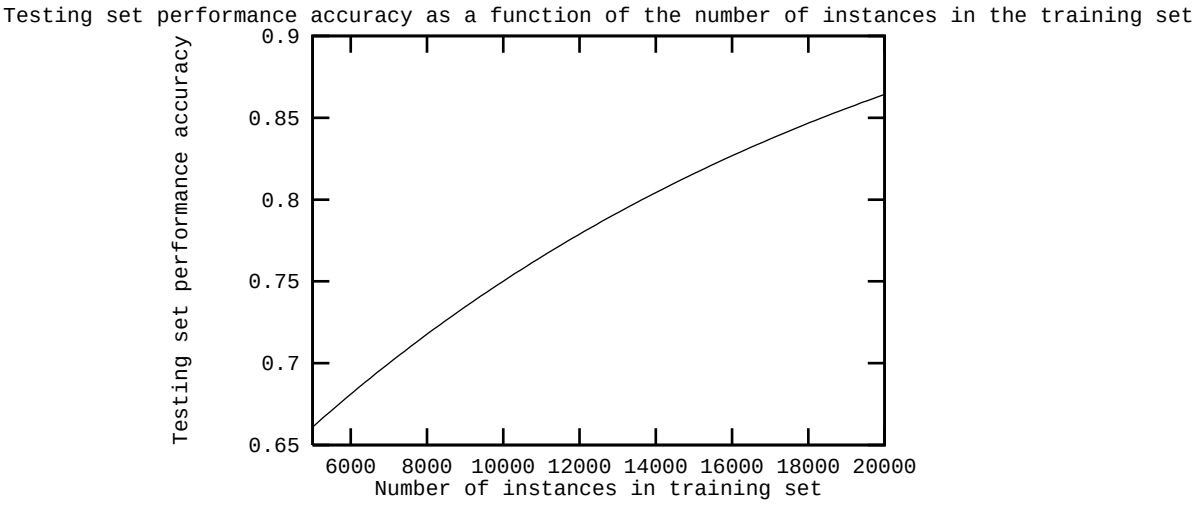

Figure 1-2: Upper bound on the training set performance accuracy as a function of the number of instances in the training set. The function, which is derived in Chapter 2, asymptotically approaches 1. As a point of comparison, the best published algorithm [Zhang et al., 1992] achieves a performance accuracy of 66.4% using a training set of approximately 17,500 instances.

learning algorithm and running it on the secondary structure prediction problem. This end result of this analysis is an equation that is a function of three variables: the number of instances in the training set, the probability that a guess is correct, and the number of *neighborhoods*. Figure 1-2 shows how the accuracy changes as a function of the number of instances in the training set. The probability that a guess is correct is set at .54 and the number of neighborhoods is set at 16384. These choices are explained in chapter 2.

The second part of chapter 3 applies PAC results to the secondary structure prediction problem. In particular, the section gives bounds on the number of instances required to learn monomials, 13-DNF formulae, and perceptrons. Table 1.1 summarizes these results which, in support of the conclusion of the first part of Chapter 2, suggest that the task of learning secondary structure to high accuracy from unannotated primary structures is not possible with current databases.

Representation	Minimum	Maximum
Monomials	2590	–
13-DNF formulae	$> 10^{30}$	$\approx 10^{35}$
Perceptrons	2730	218,214

Table 1.1: Minimal and maximal number of instances required to achieve 90% prediction accuracy with 99% confidence for three different representations. The largest secondary structure databases have approximately 20,000 instances.

1.7.2 Searching for representations that facilitate secondary structure prediction

Chapter 3 describes a database mining system that generates amino acid representations that facilitate secondary structure prediction. The main result of the chapter is a representation that captures the three properties (bulk, hydrophobicity, and charge (pI)) thought to be most important for tertiary structure prediction [Franke, 1984, Martin et al., 1989]. Although this representation is 33% shorter than the traditional representation, a neural network trained with this representation achieves the same performance accuracy as a neural network trained using the traditional representation. The representation consists of a set of 24 bitstrings each of length 16.

This database mining system is motivated by the understanding that objects in databases are not always described by features that make regularities apparent. The goal of the system described in Chapter 3 is to generate such attributes and explain why they capture patterns in the data.

The system is composed of four learning algorithms: a genetic algorithm, a neural network, a clustering algorithm, and a decision tree system. The first part of the system, composed of the genetic algorithm and the neural network, generates amino acid representations. The search for these representations is guided by the performance accuracy achieved by a neural network trained using the representations. Good representations are those that improve performance accuracy on the secondary structure prediction problem. The second part of the system, consisting of

the clustering algorithm and the decision tree system, uses a database of amino acid biochemical properties to explain why the representations generated by the first part of the system are good. This explanation is in the form of a decision tree.

Thus, the end result of running the system on secondary structure data is not only a parsimonious representation that achieves a performance accuracy equal to the traditional representation. This new representation comes with an explanation, grounded in biochemical properties, of why the representation is well suited for secondary structure prediction.

1.8 Outline of thesis

Chapter 2 develops a learning algorithm that is used to find an upper bound on the prediction accuracy of secondary structure prediction algorithms and applies results from PAC learning to the secondary structure prediction problem. Chapter 3 describes a system that takes as input the primary sequences of proteins annotated with secondary structure and produces as output an amino acid representation that facilitates secondary structure prediction. Chapter 4 summarizes the thesis and discusses future work.

Chapter 2

How many instances does a secondary structure prediction algorithm need?

The main question addressed in this chapter is: How many instances does a secondary structure prediction algorithm need to predict with high accuracy? The first section approaches this question by constructing an optimal learning algorithm and asking how well it would do on the secondary structure prediction problem. The second section applies PAC results to the secondary structure prediction problem.

2.1 Optimal learning algorithm

I give an upper bound on the performance accuracy that can be achieved on the secondary structure prediction problem by constructing an optimal learning algorithm and running it on the secondary structure prediction problem. This upper bound is a function of three parameters: the number of instances in the training set, the probability that a guess is correct, and the number of *neighborhoods*. When these parameters are set to reasonable values the upper bound is calculated to be .7502. This optimal model is very similar to one described by Quinlan [Quinlan, 1983].

The operation of the optimal learning algorithm can be understood in two parts.

When presented with an instance in the test set, the optimal algorithm can:

- Determine that the test set instance is in the same *neighborhood* as an instance in the training set. In this case, the optimal algorithm will assign the correct secondary structure to the test set instance.
- Determine that no training set instance belongs to the same neighborhood as the test set instance. In this case, the optimal algorithm guesses a secondary structure assignment.

A neighborhood is a set of instances that have similar primary sequences and that have the same secondary structure.

Why is this algorithm optimal? The first case described above sets it apart from actual algorithms. The optimal algorithm is able to determine the boundaries of a neighborhood after it has seen only one instance in that neighborhood. Actual algorithms can only crudely approximate the neighborhood boundary after seeing one instance and require many instances both inside and outside of the neighborhood to accurately approximate the boundary. Figure 2-1 illustrates the difference between the optimal algorithm and actual algorithms.

In addition, unlike current secondary structure prediction algorithms, the optimal algorithm is not forced to consider only local interactions in making its predictions. Thus, it is not constrained by bounds on the amount of information available to algorithms that only analyze local interactions [Gibrat et al., 1991].

Given this description of the optimal algorithm, it is possible to calculate its performance accuracy. Let P_{know} be the probability that a test set instance falls into the first category described above and let P_{guess} be the probability that a test set instance is correctly assigned secondary structure given that it falls into the second category. The performance accuracy, P_{total}, of the optimal algorithm is:

$$P_{total} = P_{know} + (1 - P_{know}) * P_{guess} \tag{2.1}$$

P_{know} is the probability that the test instance is in the same neighborhood as one of the training instances:

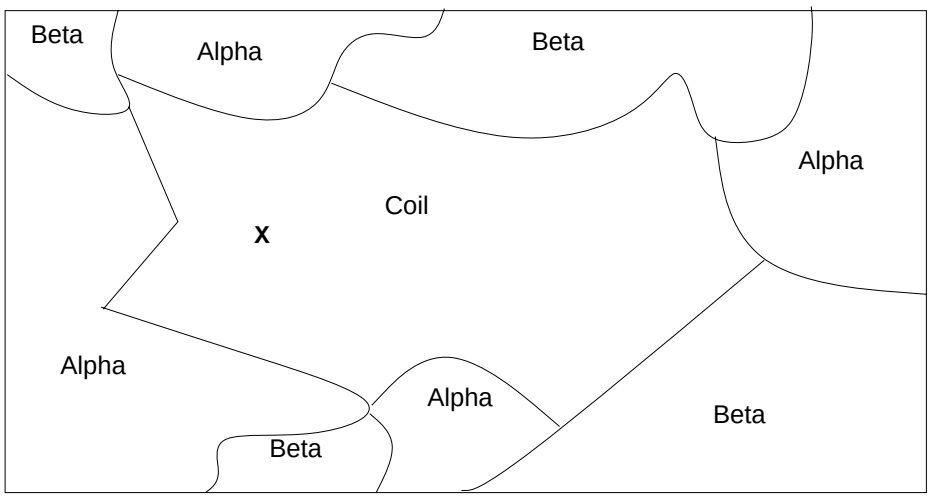

Figure 2-1: The neighborhood concept. The instance X belongs to the *coil* class. If X is in the training set, then the optimal algorithm knows what the boundaries of this *coil* neighborhood are and where they are located. It knows, for example, that the western boundary is formed by lines and that the northern boundary is formed by curves. Thus, the optimal algorithm can correctly classify all of the instances that are in the same neighborhood as X. No existing learning algorithm would be able to infer the boundaries of the neighborhood from a single instance. Although this figure is two-dimensional, in general the space of neighborhoods is multi-dimensional.

$$P_{know} = 1 - \sum_i p(N_i) * (1 - p(N_i))^M \tag{2.2}$$

where M is the number of instances in the training set, $p(N_i)$ is the probability that an instance is in neighborhood i, and i ranges over all of the neighborhoods.

We assume that P_{guess} is equal to the probability that the instance is in the *coil* class, the most frequent secondary structure class:

$$P_{guess} = p(c_{coil}) \tag{2.3}$$

where $p(c_{coil})$ is the probability that an instance belongs to the *coil* class.

Substituting into 2.1:

$$P_{total} = (1 - \sum_i p(N_i) * (1 - p(N_i))^M) + (\sum_i p(N_i) * (1 - p(N_i))^M) * p(c_{coil}) \tag{2.4}$$

To extract a number from equation 2.4 we need to assign numerical values to: M, $p(N_i)$, and $p(c_{coil})$. M, the number of instances in the training set, is approximately 10,000 [Zhang et al., 1992]. Approximately 54% of the residues are coil [Holley and Karplus, 1989, Kneller et al., 1990], so $p(c_{coil}) = .54$.

Estimating $p(N_i)$, the probability that an instance is in neighborhood i, is more difficult. I assume that instances are evenly distributed, so $p(N_i) = 1/|N|$, where $|N|$ is the number of neighborhoods. Using this approximation, 2.4 reduces to:

$$P_{total} = 1 - (1 - 1/|N|)^M + (1 - 1/|N|)^M * p(c_{coil}) \tag{2.5}$$

Now all that is needed to calculate a numerical value for P_{total} is to estimate $|N|$. Branden and Tooze [Branden and Tooze, 1991] group amino acids into four categories. Database scans have shown that there are identical pentapeptides that have different secondary structures assigned to the middle residue, but that there are no identical heptapeptides that have the same property [Kabsch and Sander, 1984, Argos, 1990]. So, an estimate for the number of neighborhoods is: $4^7 = 16384$. Of the three estimates, this one has the least support.

Substituting these choices for the three parameters into 2.5:

$$P_{total} = 1 - (1 - 1/16384)^{10000} + (1 - 1/16384)^{10000} * .54 = .7502 \tag{2.6}$$

Thus, the upper bound on secondary structure prediction accuracy, given these parameter settings, is .7502.

How sensitive is this upper bound to changes in the three parameters? Table 2.1 shows how the upper bound changes as a function of the number of instances in the training set, M; table 2.2 shows how the upper bound changes as a function of the probability that a guess will be correct, P_{guess}; and table 2.3 shows how the upper bound changes as a function of the number of neighborhoods, N.

The tables demonstrate that the upper bound on secondary structure prediction accuracy is very sensitive to the numbers assigned to the three parameters in 2.5. What does this mean? Assuming that there are no egregious errors in this analysis,

M	P_{guess}	$\|N\|$	prediction accuracy
0	.54	16384	.5400
1000	.54	16384	.5672
5000	.54	16384	.6610
10000	.54	16384	.7502
20000	.54	16384	.8643
30000	.54	16384	.9263
50000	.54	16384	.9783
100000	.54	16384	.9990

Table 2.1: Prediction accuracy as a function of M, the number of instances in the training set. The fourth line is the case which is described in the text. Using current databases it may be possible to have 20,000 instances in the training set. In this case the upper bound increases to .8643. In the first line the prediction accuracy of .54 is entirely due to guesses.

P_{guess}	M	$\|N\|$	prediction accuracy
0.00	10000	16384	0.4569
0.25	10000	16384	0.5926
0.45	10000	16384	0.7013
0.50	10000	16384	0.7284
0.54	10000	16384	0.7502
0.55	10000	16384	0.7556
0.60	10000	16384	0.7827
0.75	10000	16384	0.8642
1.00	10000	16384	1.0000

Table 2.2: Prediction accuracy as a function of P_{guess}, the probability that a guess will be correct. The analysis in the text corresponds to the fifth line. The first line shows that if guesses were always wrong then the performance accuracy would be .4569. Thus, the contribution of guesses to the total prediction accuracy is the difference between the total prediction accuracy and .4569. So, for the parameter set in the fifth line, the contribution of guesses to the total prediction accuracy is: $.7502 - .4569 = .2933$.

$\|N\|$	M	P_{guess}	prediction accuracy
1	10000	.54	1.0000
5000	10000	.54	0.9378
10000	10000	.54	0.8308
15000	10000	.54	0.7638
16384	10000	.54	0.7502
18000	10000	.54	0.7361
20000	10000	.54	0.7210
40000	10000	.54	0.6418
80000	10000	.54	0.5941
200000	10000	.54	0.5624
1000000	10000	.54	0.5446

Table 2.3: Prediction accuracy as a function of $|N|$, the number of neighborhoods. The fifth line corresponds to the analysis in the text. As the number of neighborhoods increases, the prediction accuracy converges to P_{guess}.

two conclusions suggest themselves. First, the optimal learning algorithm model is not accurate. Either it fails to capture the essence of learning algorithms or the essence of the secondary structure prediction problem (or both). Second, the performance of algorithms on the secondary structure prediction problem actually does depend critically on the number of training instances, the probability of guessing correctly, and the number of neighborhoods.

Which of these two conclusions is correct? Some evidence indicates that the performance of learning algorithms varies with the number of instances in the training set [Qian and Sejnowski, 1988]. This data can be checked against Table 2.1 to see if the optimal learning model accurately describes the behavior of actual learning algorithms. The performance accuracy's dependence on the probability of guessing correctly can be examined in the same way. Dependence on the number of neighborhoods can be determined by first testing the performance of actual learning algorithms on an array of artificial problems that fix the number of neighborhoods and then comparing this performance to the optimal model's performance. If the optimal model passes all of these tests, then the second conclusion would be more likely to be correct than the first.

2.2 PAC results

This section applies PAC results to the secondary structure prediction problem.

We consider several restrictions on the answer to the secondary structure prediction problem and on the nature of the learning algorithms used to address the problem. For example, if the assumption is made that the set of all alpha helices can be differentiated from all other secondary structure classes using monomials[1], then learning a monomial representation for alpha helices that separates it from other secondary structure classes is shown to require a small number of instances.

This section does not present new algorithms or new analytical results. Rather, it applies existing results in theoretical machine learning to the secondary structure prediction problem.

Part 1 informally introduces the probably approximately correct (PAC) model which will be the general theoretical framework used in this section to give upper and lower bounds on the number of instances required to learn secondary structure. Part 2 discusses the "thirding" algorithm which makes no assumptions about the solution to the secondary structure problem and, therefore, gives very weak results. In light of these weak results, Part 3 restricts the solution to the secondary structure problem and describes stronger results.

2.2.1 The PAC model

This section uses a general theoretical framework, called the probably approximately correct (PAC) model, for analyzing algorithms that learn concepts from examples [Valiant, 1984]. The main idea of this framework is that a learning algorithm, after processing a certain number of instances, should produce with high probability, a hypothesis that makes predictions that, with high probability, are the same as those made by the the correct hypothesis.

Valiant [Valiant, 1984] and Kearns and Varizani [Kearns and Vazirani, 1992] both

[1] A monomial is a finite conjunction of literals. For example, $x_1 \wedge x_2 \wedge \overline{x_3}$ is a monomial with three literals.

give formal descriptions of PAC learning. Here I give an informal description. Let H_1, H_2, ... be a countable set of subsets of a countable instance space. The task is to identify one of these subsets, H_*. The learning algorithm outputs a hypothesis, H_{answer}, that is close to H_* with high probability:

$$\Pr[d(H_*, H_{answer}) \geq \epsilon] \leq \delta$$

where d is a function that returns the probability that an instance chosen from the instance space according to an unknown probability distribution is in one but not the other of H_* and H_{answer}.

Intuitively, this formula says that with probability at least $1 - \delta$ the difference between H_* and H_{answer} will be less than ϵ. For example, if we want to be 99% sure that H_{answer} is 90% accurate (i.e. it is within 10% of the correct concept, H_*) then $\epsilon = .1$ and $\delta = .01$.

2.2.2 No assumptions: The thirding algorithm

The thirding algorithm makes no assumptions about the nature of the solution to the secondary structure prediction problem and assumes virtually unlimited computational power. This algorithm serves to highlight the advantages of the algorithms that will be described in the next section.

The thirding algorithm maintains a set of hypothesized solutions all of which are consistent with all of the training instances it has processed. When it processes a new training instance, the algorithm uses its current set of hypotheses to classify the instance into the alpha, beta, or coil class. If the classification is incorrect, then at least one third of the hypothesis are eliminated, hence the algorithm's name. If the classification is correct, then all of the hypotheses that incorrectly classified the new instance are eliminated.

Specifically, a hypothesis is a set of three lists that correspond to the alpha, beta, and coil classes. Each element in a list is simply an instance and each possible instance appears in exactly one list. Thus, the union of the three lists contains all possible

instances. A training set instance is classified by the list in which it appears. For example, if the element EGDAAKGE is in the coil list then the instance EGDAAKGE is classified into the coil class.

The thirding algorithm classifies a new training set instance by using the classifications of the hypotheses. A simple pigeonhole argument shows that if all of the classifications are tallied at least one of the three secondary structure classes will have been predicted by at least one third of the hypotheses. This consensus classification is the prediction of the thirding algorithm. If the prediction is incorrect, then at least one third of the hypotheses are inconsistent with the training data and can be eliminated. If the prediction is correct, all of the hypotheses that did not agree with the consensus prediction can be eliminated.

The thirding algorithm makes the most guaranteed progress when it fails to correctly classify a training set instance. If there are n hypotheses in the original set then it needs to process $\log_{3/2} n = (\log_2 n)/(\log_2 3/2)$ instances that it classifies incorrectly to guarantee that the set of hypotheses is narrowed down to just a single consistent hypothesis.

How large is $\log_{3/2} n$? If the initial hypothesis set consists of all possible hypotheses and each instance processed by the algorithm is unique, then $\log_{3/2} n$ is equal to the total number of instances. Of course, this is not a helpful practical result. However, if the original hypothesis set is restricted, then the number of training set instances may be substantially decreased. The hypotheses should be restricted in a way that captures some underlying property of the solution to the secondary structure prediction problem. So, for example, reducing the set of hypotheses by forcing each hypothesis to consider tryptophan and glycine to be equivalent is not satisfactory. The next section considers different ways of restricting the set of hypotheses and gives bounds on the number of training instances required to find hypotheses that with high confidence are close to the correct concept description.

2.2.3 Restricting the set of hypotheses

This section considers three restrictions on the set of hypotheses. The first two parts assume that the hypotheses can be represented as particular kinds of boolean formulae, and the third part assumes that the hypotheses can be captured by perceptrons with step thresholds.

Monomials

First we restrict each hypothesis to consist of three monomials, one each for the alpha, beta, and coil class. There are twenty variables for each position, each one corresponding to one of the amino acids. Variable x_i is true when the amino acid at position i is x and false otherwise.

How might these three monomials be used to describe a solution to the secondary structure prediction problem? Suppose that the monomial that corresponds to the alpha class is: $G_1 \wedge R_2 \wedge \overline{F_5}$. This monomial is true exactly when the instance has glycine as the first amino acid, arginine as the second amino acid, and does not have phenylalanine as the fifth amino acid. Notice that there are many concepts that cannot be expressed by these monomials. For example, it is impossible to assign all instances that have leucine or isoleucine as the first amino acid to the alpha class.

Ehrenfeucht *et al.*[Ehrenfeucht et al., 1988] show that the minimal number of instances needed to learn a monomial is

$$\max\left(\frac{1}{2\epsilon}\ln\frac{1}{\delta}, \frac{n-1}{\epsilon}\right)$$

where n is the number of variables, $0 < \epsilon \leq \frac{1}{8}$, and $0 < \delta \leq \frac{1}{100}$.

To extract a number from this equation we need to assign numbers to ϵ, δ, and n. Let $\epsilon = .1$ and $\delta = .01$. This parameter setting means that we want to be at least 99% confident that the algorithm has a performance accuracy of at least 90%.

The number of literals is simply $20 * l$ where l is the length of an instance. If $l = 13$ there are 260 literals, and the minimum number of instances needed to learn the monomial is $\max(\frac{1}{.2}\ln(\frac{1}{.01}), \frac{259}{.1}) = 2590$. This lower bound is tight to within a

constant since Valiant [Valiant, 1984] describes an $O(\frac{1}{\epsilon}\ln(\frac{1}{\delta}) + \frac{n}{\epsilon})$ algorithm for learning monomials. The largest databases used in secondary structure prediction to date have approximately 20,000 instances [Zhang et al., 1992] and so learning monomials of this form is well within the realm of possibility. Unfortunately, the class of concepts that can be represented by these monomials is very restricted.

So to increase the number of concepts that can be described, suppose that we increase the number of variables by allowing them to represent not just individual amino acids but pairs of amino acids. This allows the monomial to, for example, represent the class of all instances that have a leucine or isoleucine as the first amino acid. There are $l * 20 * 19/2 = 190 * l$ of these literals where l is again the length of the instance. For $l = 13$ there are 2470 literals of this type in addition to the original 260 literals, for a total of $2470 + 260 = 2730$ literals. Using the same parameter settings as before, the number of instances needed to learn the monomial is at least $\max(\frac{1}{.2}\ln(\frac{1}{.01}), \frac{2729}{.1}) = 27290$. Thus, what appears to be a small increase in the representational power of the monomial leads to an order of magnitude increase in the number of instances required to learn it.

Disjunctive normal form

Now we restrict each hypothesis to consist of three boolean formulae in disjunctive normal form. A formula in disjunctive normal form is a finite disjunction of conjunctions: $T_1 \vee T_2 \vee ... \vee T_n$ where T_i is a monomial. Any boolean formula can be expressed in disjunctive normal form (DNF). We use this fact to derive an upper bound on the number of instances needed to solve the secondary structure prediction problem.

A kDNF formula is a DNF formula in which the monomials are of length at most k. We fix the number of variables to be 260 and interpret them as described above. Given this set of variables any monomial with more than l unique literals would be contradictory, where l is the length of instance. If $l = 13$ then k can be set to 13 without restricting the power of the boolean formulae. Thus, for this choice of variables, every DNF formula can be represented as some 13-DNF formula. Furthermore, every possible subset of instances of length 13 can be represented using

a 13-DNF formula.

Combining the results in Blumer *et al.*[Blumer et al., 1986] and in Ehrenfeucht *et al.*[Ehrenfeucht et al., 1988] the upper bound on the number of instances needed to learn a 13-DNF formula with 260 variables is:

$$\max\left(\frac{4}{\epsilon}\ln\frac{2}{\delta}, \frac{8*(260^{13}-1)}{\epsilon}\ln\frac{8*(260^{13}-1)}{\epsilon}\right)$$

where the notation is defined as above.

Setting $\epsilon = .1$ and $\delta = .01$ as before, the maximum number of instances needed is approximately $1.5*10^{35}$. To learn such a 13-DNF formula for each of the three types of secondary structures would require at most $4.5*10^{35}$ instances. Several researchers (e.g., [Qian and Sejnowski, 1988, Holley and Karplus, 1989, Zhang et al., 1992]) have informally suggested that there is an upper bound between 60% and 80% on the performance accuracy that can be achieved using local information. Assuming that alpha helices, beta sheets, and coils can each be represented by using a 13-DNF formula such that there is no instance for which more than one of the formulas is true, then this result shows that these informal arguments are incorrect. In particular, this result shows that at most approximately $4.5*10^{35}$ instances need to be processed to be at least 99% confident that the hypothesis is at least 90% accurate.

The 13-DNF representation can also be used to give lower bounds on the number of instances. As with monomials, the best algorithm, described in [Valiant, 1984], is within a constant factor of this lower bound which is in turn only a log factor less than the upper bound given above. Thus, the lower bound on the number of instances needed to learn a 13-DNF formula is, for all practical purposes, prohibitive.

However, the number of instances can be reduced by decreasing the number of literals in a monomial. What is the justification for doing so? Database searches show that there are identical pentapeptides that have different secondary structures assigned to the middle residue, but that there are no identical heptapeptides that have the same property [Kabsch and Sander, 1984, Argos, 1990]. If this property holds for all proteins and not just those that exist in current databases, then 7-DNF formulae

can be used to describe secondary structures. Although this reduces the number of instances needed to about 10^{20}, far too many instances are still required.

The lower bound for PAC learning k-Decision lists [Rivest, 1987] is identical to the one for learning k-DNF formulas. A decision list is a finite sequence (T_1, b_1),...,(T_l,b_l) where T_i is a monomial of at most k literals and b_i is a boolean value. The value of the decision list is b_i where T_i is the first monomial that is true of the instance. Although the lower bound is the same, k-Decision lists can describe a set of concepts that is a proper superset of the set of concepts that can be expressed by k-DNF formulae.

Perceptrons

Perceptron learning algorithms with sigmoid threshold functions are the most widely used programs for secondary structure prediction [Qian and Sejnowski, 1988, Holley and Karplus, 1989, Zhang et al., 1992], although other techniques, such as nearest neighbor approaches [Salzberg and Cost, 1992], are now becoming popular. In this section we briefly explore the number of instances needed to train a perceptron with stair-step thresholds.

Once again we consider a hypothesis to be a set of three perceptrons, one for each of the three types of secondary structure. To train a single perceptron with n units requires at least

$$\max\left(\frac{1}{2\epsilon}\ln\frac{1}{\delta}, \frac{n}{\epsilon}\right)$$

instances and at most

$$\max\left(\frac{4}{\epsilon}\ln\frac{2}{\delta}, \frac{(8*n)}{\epsilon}\ln\frac{(8*n)}{\epsilon}\right)$$

instances.

When perceptrons are used to learn secondary structure each amino acid is typically represented by a bit string of length 21 (one bit for each amino acid and an additional wrap-around bit that is used to pad instances that are near the ends of a

protein). Thus, if the instances are of length 13, then the number of input units is $13 * 21 = 273$. For $\epsilon = .1$ and $\delta = .01$, this gives a lower bound of 2730 instances and an upper bound of 218,214 instances. If ϵ is increased to .2 then the upper bound on the number of instances falls to 101,538 which is only a factor of five greater than the number of instances available from current databases. To train three perceptrons would require at most $3 * 101,538 = 304,614$ instances.

Chapter 3

Searching for representations that facilitate secondary structure prediction

This chapter describes a system that generates good amino acid representations and that explains why they are good. First, the best 12-bit representation discovered by the system is used to demonstrate the functions and capabilities of the system. Second, the four subsystems that constitute the system are described in detail. Third, the best 16-bit representation, which is the best representation found to date, is discussed. Finally, results of several control experiments are presented.

3.1 Overview

The goal of the system shown in Figure 1-1 is to produce amino acid representations that facilitate secondary structure prediction. The system is divided into four subsystems:

- A search algorithm that searches over the space of representations. I use a genetic algorithm that searches over the space of bit strings.

- A learning algorithm that quantifies the quality of a representation. In this work, the quality of a representation is the performance accuracy of a neural network trained using that representation.

- A clustering algorithm that groups amino acids using their representations. I use a clustering algorithm that uses Hamming distance to group amino acids.

- A learning algorithm that explains these clusterings using biochemical data. I use a decision tree system that predicts the cluster of an amino acid given its biochemical properties.

These four subsystems are grouped into two parts. The first part, which consists of the genetic algorithm and the neural network, produces amino acid representations that are designed to improve secondary structure prediction. These amino acid representations are composed of 24 bitstrings. There is one bitstring for each amino acid and an additional four bitstrings to represent three characters that appear in the primary sequence database (B for asparagine or aspartic acid, X for unknown, and Z for glutamine or glutamic acid) and the wrap-around character. The traditional orthogonal amino acid representation, which has 24 bits per bitstring[1], is shown in Figure 3-1. A 12-bit representation generated by the system is shown in Figure 3-2.

The second part, composed of the clustering algorithm and the decision tree system, explains the representations generated by the first part in terms of biochemical properties of amino acids. Figure 3-3 shows a clustering of the 12-bit representation presented in Figure 3-2. Figure 3-4 is a decision tree created from this clustering and the biochemical properties database shown in Table 3.1.

The particular choice of a neural network and genetic algorithm for the first component of the system and the choice of a clustering algorithm and a decision tree system for the second part is not essential. What is important is that the first part produces amino acid representations that attempt to optimize some metric (such as prediction accuracy) and that the second part explains why these representations

[1]Heretofore called a 24-bit representation.

```
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     Wrap-around
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     Alanine
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     Asparagine or aspartic acid
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     Cysteine
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     Aspartic acid
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     Glutamic acid
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     Phenylalanine
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     Glycine
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     Histidine
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0     Isoleucine
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0     Lysine
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0     Leucine
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0     Methionine
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0     Asparagine
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0     Proline
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0     Glutamine
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0     Arginine
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0     Serine
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0     Threonine
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0     Valine
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0     Tryptophan
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0     Unknown
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0     Tyrosine
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1     Glutamine or glutamic acid
```

Figure 3-1: The traditional orthogonal representation. Twenty four bits are used to represent the twenty three characters that appear in the primary structure of proteins (the 24th character is the wrap-around character which is an anomaly produced by the way in which the primary structure is preprocessed; see the text).

```
0 1 1 1 1 0 0 1 1 1 0 0     Wrap-around
0 1 0 0 0 0 1 0 1 0 1 1     Alanine
0 0 0 0 1 0 1 1 0 0 1 0     Asparagine or aspartic aci
0 0 0 0 0 1 0 1 0 1 0 0     Cysteine
1 0 1 0 1 0 0 0 0 0 0 0     Aspartic acid
0 1 0 0 0 0 1 0 1 1 1 1     Glutamic acid
1 1 0 0 1 1 1 0 0 1 1 0     Phenylalanine
1 1 1 0 0 0 0 0 1 1 1 0     Glycine
0 0 1 0 1 1 1 1 0 1 0 0     Histidine
0 0 1 1 1 1 1 1 1 0 0 1     Isoleucine
1 1 1 0 1 1 0 1 1 1 0 1     Lysine
0 1 1 0 1 0 1 0 0 1 1 0     Leucine
1 0 0 1 1 1 1 0 0 1 0 0     Methionine
0 1 1 1 1 0 1 1 1 1 0 0     Asparagine
1 1 0 1 0 0 0 1 0 0 0 0     Proline
1 1 1 0 0 0 1 1 1 1 1 1     Glutamine
0 1 1 1 0 0 1 1 1 1 0 0     Arginine
0 0 1 1 1 0 1 1 1 1 1 0     Serine
0 0 1 1 1 0 0 0 0 0 1 0     Threonine
1 1 1 1 0 1 1 0 1 0 0 0     Valine
0 0 0 1 1 0 0 0 1 1 1 1     Tryptophan
1 0 0 0 1 1 0 0 1 1 0 0     Unknown
1 0 0 1 1 0 1 0 1 0 1 1     Tyrosine
0 0 1 0 0 0 0 0 0 0 1 0     Glutamine or glutamic acid
```

Figure 3-2: A 12-bit representation generated by the system.

```
Concept hierarchy is:
N2
   N32
      TYROSINE
      TRYPTOPHAN
   LYSINE
   N23
      SERINE
      ARGININE
      ASPARAGINE
   N28
      VALINE
      ISOLEUCINE
   N18
      PROLINE
      METHIONINE
      THREONINE
      ASPARTIC-ACID
      HISTIDINE
      CYSTEINE
   N9
      LEUCINE
      GLYCINE
      PHENYLALANINE
   N5
      GLUTAMINE
      GLUTAMIC-ACID
      ALANINE
```

Figure 3-3: Clustering of the 12-bit representation shown in Figure 3-2. All of the branches below depth 2 have been folded into their parents.

Hyd	Cha	Pol	Ali	Aro	Sul	Bas	Aci	pI	Hyd2	Bulk	Abbr
yes	no	no	yes	no	no	no	no	6.02	0.32	-1.44	ALA
no	yes	no	no	no	no	yes	no	10.76	-1.07	1.16	ARG
no	no	yes	no	no	no	no	yes	5.41	-.96	-.34	ASN
no	yes	no	no	no	no	no	yes	2.97	-1.07	-.54	ASP
no	no	yes	no	no	yes	no	no	5.02	1.5	-0.75	CYS
no	yes	no	no	no	no	no	yes	3.22	-1.03	.17	GLU
no	no	yes	no	no	no	no	yes	5.65	-1.05	0.22	GLN
no	no	no	yes	no	no	no	no	5.97	-.03	-2.16	GLY
no	no	yes	no	no	no	yes	no	7.58	-.13	.52	HIS
yes	no	no	yes	no	no	no	no	5.98	1.52	.21	ILE
yes	no	no	yes	no	no	no	no	5.98	1.14	.25	LEU
no	yes	no	no	no	no	yes	no	9.74	-1.76	.68	LYS
yes	no	no	no	no	yes	no	no	5.75	1	.44	MET
yes	no	no	no	yes	no	no	no	5.98	1.16	1.09	PHE
yes	no	no	no	no	no	no	no	6.1	-.72	-.71	PRO
no	no	yes	no	no	no	no	no	5.68	-.46	-1.21	SER
no	no	yes	no	no	no	no	no	6.53	-.36	-.67	THR
no	no	yes	no	yes	no	no	no	5.88	0.67	2.08	TRP
no	no	yes	no	yes	no	no	no	5.65	-.07	1.34	TYR
yes	no	no	yes	no	no	no	no	5.97	1.38	-.34	VAL

Table 3.1: Database of biochemical properties of amino acids. The decision tree system uses this database to explain the clustering of amino acids. The first eight properties are qualitative, binary properties, while the last three properties are quantitative, numerical properties: Hyd = hydrophobic, Cha = charged, Pol = polar, Ali = aliphatic, Aro = aromatic, Sul = sulfur, Bas = basic, Aci = acidic, pI = pI value, Hyd2 = hydrophobicity scale, Bul = measure of bulk, and Abbr = three letter amino acid abbreviation. The Hyd, Cha, and Pol attributes are from [Branden and Tooze, 1991]; the Ali, Aro, Sul, Bas, and Aci attributes are from [Stryer, 1988]; the pI attribute is from [Mahler, 1971]; and the Hyd2 and Bulk attributes are from [Kidera et al., 1985].

```
pi > 7.58
pi <= 7.58
|   bulk > 1.16
|   bulk <= 1.16
|   |   aliphatic = yes
|   |   aliphatic = no
|   |   |   hydrophobic2 <= -1.03
|   |   |   hydrophobic2 > -1.03
```

Figure 3-4: A decision tree created from the clustering shown in Figure 3-3 and the data shown in Figure 3.1.

are good in terms of some independent qualities (such as amino acid biochemical properties).

For example, the neural network could be replaced by any classification scheme, such as a nearest-neighbor method [Aha et al., 1991], and the genetic algorithm could be replaced by any search algorithm, such as a simulated annealing procedure [Kirkpatrick et al., 1983], and the essential structure of the system would remain the same. This essential structure and the particular choices made for this system are shown in Figure 3-5.

We now turn to a detailed description of the system.

3.2 System description

The structure of this description mirrors the structure of the system itself. This section is divided into two parts. The first describes how the system generates amino acid representations that facilitate secondary structure prediction, and the second describes how the system explains these representations in terms of the biochemical properties of amino acids.

3.2.1 Generating amino acid representations

The first part of the system is composed of a genetic algorithm and a neural network.

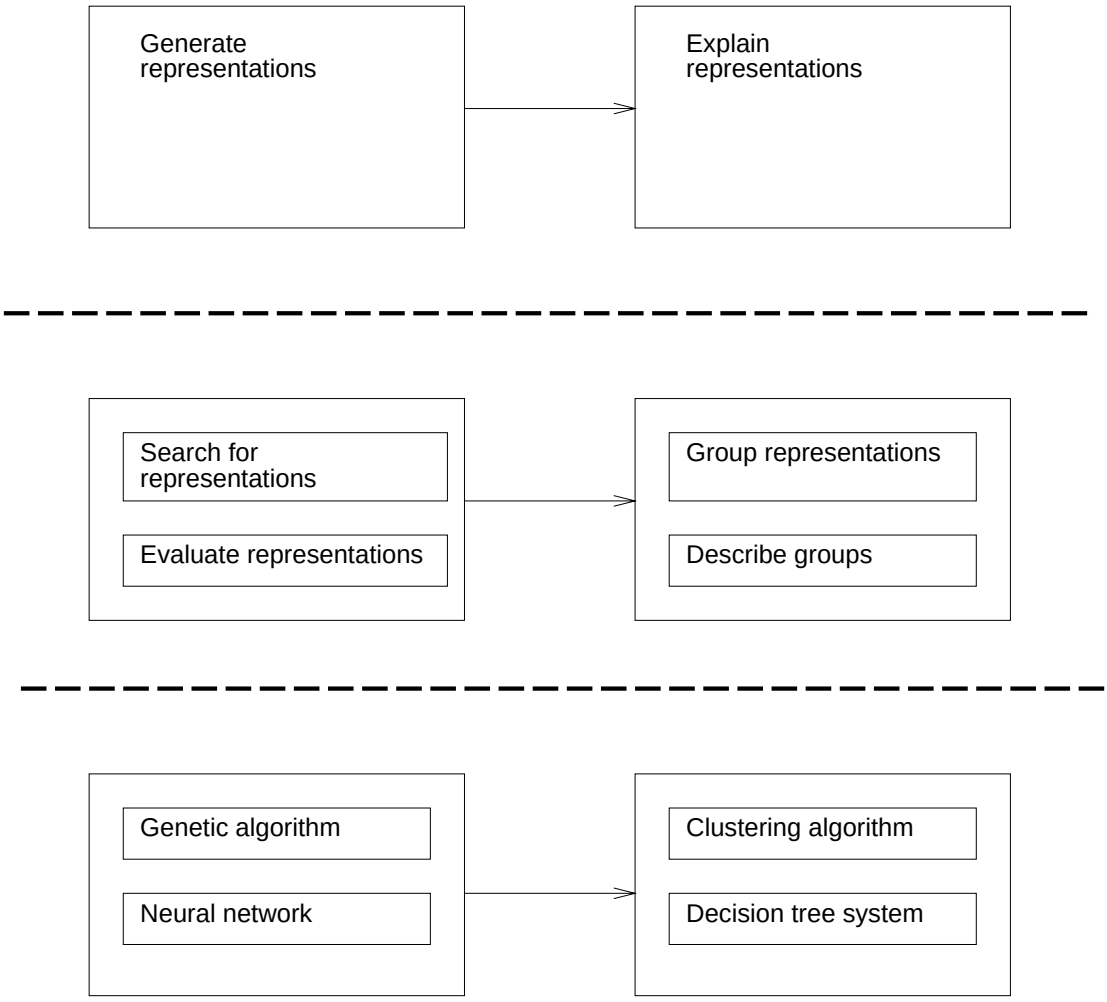

Figure 3-5: The essential structure of the system and the particular choices made for the work described in this chapter.

```
t = 0
Initialize P(t)
While t < GENERATION
        Assign fitnesses to the individuals in P(t)
        Select P(t+1) from P(t)
        Apply mutation and crossover operators to P(t+1)
        t = t + 1
```

Figure 3-6: Genetic algorithm pseudocode. In this work, each individual in P(t) is a bitstring. This bitstring is of size 24^*l, where l is the number of bits used to represent a single amino acid. The traditional representation uses 24 bits per amino acids. GENERATION is a parameter that is set by the user.

Genetic algorithm

Genetic algorithms are patterned after biological systems. They maintain a population of individuals that undergo crossover and mutation (recombination). Individuals that are significantly less fit than the rest of the population are gradually eliminated, while stronger individuals are propagated (selection). One round of selection and recombination is called a generation. In this way, the genetic algorithm evolves a population of individuals of increasingly higher fitness. Pseudocode for a genetic algorithm is shown in Figure 3-6.

In this system individuals are amino acid representations (24 bitstrings). The system begins with a set of randomly generated representations and uses the crossover and mutation operators to improve them. To evaluate the quality of each representation a neural network is trained to predict secondary structure using the representation. The fitness of the representation is simply the performance accuracy of the neural network on a set of test instances.

The crossover operator selects two individuals in the population and exchanges their genetic material to produce two new individuals. A diagram of this operator is shown in Figure 3-7.

The mutation operator randomly selects a bit in an individual and changes it (from a "0" to a "1" or from a "1" to a "0"). A diagram of this operator is shown in Figure 3-8.

Genetic algorithms have several parameters that can be set:

Individual 1 = 010100010101:000101000111

Individual 2 = 101010010101:010101000101

Offspring 1 = 010100010101:010101000101

Offspring 2 = 101010010101:000101000111

Figure 3-7: The genetic algorithm crossover operator. Two individuals are chosen from the population, a crossover point is randomly selected (indicated by the ":"), and two offspring are produced. The first offspring is the result of concatenating the bitstring to the left of the crossover point in individual 1 with the bitstring to the right of the crossover point in individual 2. The second individual is created by concatenating the bitstring to the right of the crossover point in individual 2 with the bitstring to the right of the crossover point in individual 1.

Individual = 0101010101000110100:0110

New individual = 0101010101000110101:0110

Figure 3-8: The genetic algorithm mutation operator. The bit to the left of the ":" is changed from a "0" to a "1". The bit that is mutated is chosen randomly.

- The expected number of crossovers for each individual.
- The expected number of mutations for each bit.
- The number of individuals in a population.
- The number of generations.
- The selection procedure.

In this work, the expected number of crossovers for each individual was set to .7, and the expected number of mutations per bit was set to .00042. These numbers are similar to the ones typically used by researchers in the field. Both of them are the default settings of one of the public domain genetic algorithm packages I used. The mutation rate is quite low. The number of bits in the orthogonal representation is $24 * 24 = 576$, so, on average, $576 * .00042 \approx .24$ bits will be mutated per individual per generation. In some experiments the mutation rate was increased to .1. This change did not affect the performance accuracy, thus confirming other experiments with genetic algorithms which show that their performance is robust with respect to these parameters.

The number of individuals in a population and the number of generations was, unfortunately, constrained by the available computational resources. A typical genetic algorithm has fifty individuals and runs for two hundred generations. In most of the experiments I describe, a genetic algorithm with ten individuals was run for ten generations. The 16-bit representation described below was produced by a genetic algorithm with twenty individuals that ran for twenty-five generations. Over the course of these genetic algorithm runs, the performance accuracy of the best individual improves by approximately 2%.

The selection procedure uses the fitnesses of the individuals to compute the expected number of copies of each individual that will participate in the next round of recombination. The standard selection procedure, called proportional selection, divides the fitness of an individual by the average fitness of the individuals in the population to arrive at the expected number of copies of that individual that will

Crossover rate	0.7
Mutation rate	0.00042
Individuals	10
Generations	10
Selection procedure	Proportional

Table 3.2: Genetic algorithm parameter settings.

participate in the next round of recombination. So, for example, if the average fitness of a population is 8 and the fitness of a particular individual is 16, two copies of this individual will, on average, participate in the next recombination step which leads to the creation of the next population.

All of the genetic algorithms described here used proportional selection. The setting for the five parameters are summarized in Table 3.2.

Two public domain genetic algorithm packages, GAucsd 1.4 [Schraudolph and Grefenstette, 1992] and GENEsYs 1.0 [Bäck and Hoffmeister, 1992], were used in this work. Appendix A discusses GENEsYs in detail.

Neural network

Neural networks have been widely used for function approximation and classification. Here, a perceptron (a neural network with no hidden units) with sigmoid units is used to learn a function that maps primary structure to secondary structure.

The neural network is trained using a database of proteins identical to the one described by Zhang *et al.*[Zhang et al., 1992]. Table 3-9 lists some of the characteristics of this database.

The neural network does not process an entire protein at one time. Instead, each protein is divided into pieces, called "windows", each of which has thirteen residues. At the ends of the protein these windows are padded with the wrap-around character mentioned above. The neural network has three outputs, one for each of the three possible secondary structure classes (alpha helix, beta sheet, and coil). The neural network is trained to predict the secondary structure class of the middle amino acid

Number of proteins	113
Number of residues	19,861
Average number of residues per protein	176
Percentage of residues that are in the coil class	52.3
Percentage of residues that are in the beta sheet class	20.8
Percentage of residues that are in the alpha helix class	27.0

Figure 3-9: Characteristics of amino acid database. The percentages of residues in the coil, beta sheet, and alpha helix classes do not sum to 100% because of roundoff errors.

in the window. The structure of the neural network is depicted in Figure 3-10.

Furthermore, each amino acid is encoded using a bitstring. The traditional orthogonal representation represents each amino acid with 24 bits, so the typical perceptron has $13 * 24 = 312$ input units. Since each unit is connected to three output units, there are a total of $312 * 3 = 936$ weights that needed to be updated at each epoch.

The neural network is trained using a database of 48 proteins and tested using a database of 65 proteins.[2] The training set is divided into two parts, one of which is used to prevent overfitting. The neural network is trained for 200 epochs. This training period was chosen because it has been used successfully by Zhang *et al.*[Zhang et al., 1992]. The training scheme is depicted in Figure 3-11.

Training a neural network of this size on such a large database takes approximately twenty minutes on a SparcStation 10. The 16-bit representation discussed below was generated by a run that took approximately 25 Cray C90 CPU hours.

Three neural network parameters, the learning rate, the inertia, and the number of hidden units, were set using coarse-grained searches.

The learning rate is a multiplicative factor that helps determine how much the weights change. A high learning rate causes the neural network to take large jumps in weight space. Neural networks with learning rates of 5, 1, 0.005, 0.001, and 0.0001 were trained using the traditional orthogonal representation. The neural networks with learning rates of 0.001 had the highest accuracies, so the learning rate was set

[2]These numbers were chosen because 48 is evenly divisible by 3 and 4 and 65 is evenly divisible by 5.

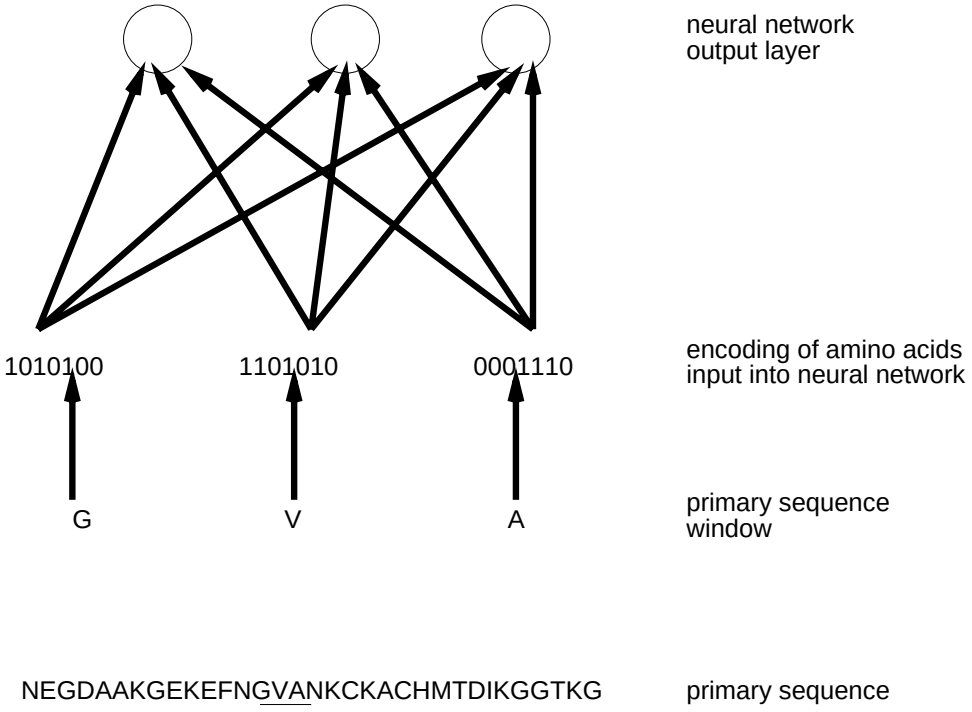

Figure 3-10: Neural network structure. The primary sequence of a protein is divided into windows and each amino acid is then encoded using the bitstrings in the amino acid representation. The bitstring representation is the input into the neural network which has no hidden layers. The three outputs correspond to the three types of secondary structure.

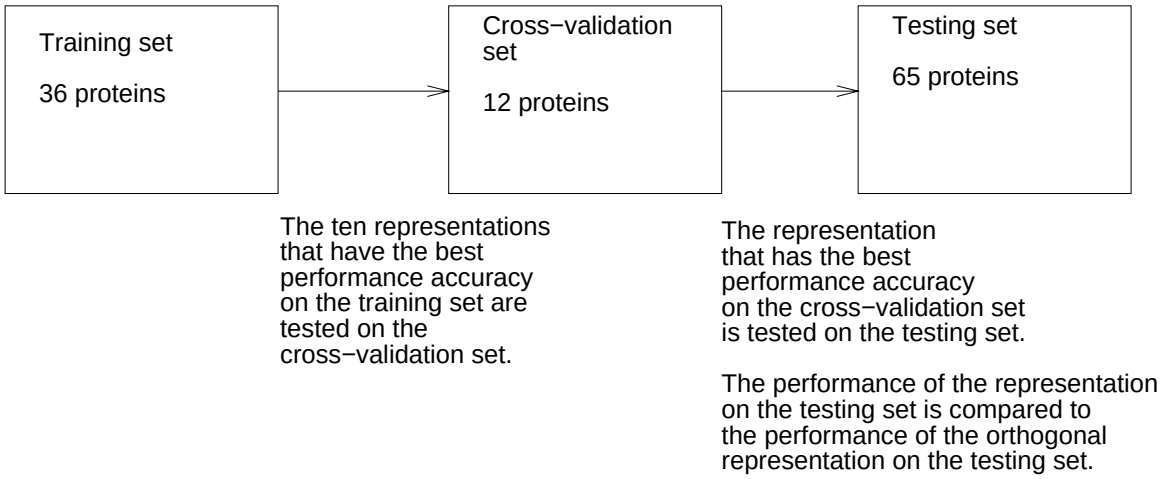

Figure 3-11: Neural network training scheme. The genetic algorithm uses the performance accuracy on the training set as a measure of the quality of the representation. The cross-validation set is used to eliminate representations that have overfitted to the training set. The performance accuracy on the testing set is used to compare the representations generated by this system to the traditional orthogonal representation.

Learning rate	0.001
Inertia	0.05
Hidden units	0

Table 3.3: Neural network parameters settings.

at 0.001 for the rest of the experiments.

The inertia is a parameter that determines how the last weight update affects the current weight update. If the inertia is high then most of the change in the weight is determined by the last weight update. This parameter prevents the neural network from changing direction drastically in weight space. Neural networks with inertias of 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, and 0.9 were trained, and a network with an inertia of 0.5, which equally weights the last update and the current weight update, was found to have the best accuracy.

This coarse-grained parameter scanning also was performed to determine the number of hidden units. Not all possible combinations of learning rate, inertia, and number of hidden units were tried. Instead, a few points in this space were chosen and the results from these experiments were used to determine the next settings. I do not know of another way to set these parameters. The neural network parameters are summarized in Table 3.3.

The neural network used in this system was implemented in ASPIRIN [Leighton, 1992], a public domain program.

3.2.2 Explaining amino acid representations

The second part of the system is composed of a clustering algorithm and a decision tree system.

Clustering algorithm

The clustering algorithm used in this thesis is COBWEB [Fisher, 1987] as implemented by a public domain package written by McKusick and Thompson [McKusick and Thompson, 19

COBWEB is an incremental clustering algorithm that produces a concept hierarchy. When processing a new instance COBWEB employs an information theoretic metric to decide whether to split a node in the concept hierarchy, merge two nodes in the concept hierarchy, create a new node in the concept hierarchy, or add the new instance to a current node in the concept hierarchy.

Here COBWEB clusters amino acids using the bitstrings generated by the first part of the system. The first instance is placed at the root node of the concept hierarchy. Each node summarizes a set of instances. The root node summarizes all of the instances while nodes at the bottom of the tree summarize a relatively small number of instances. Each node contains two parts:

- The probability that an instance is described by that node. This is computed by dividing the number of instances by the total number of instances described by that node. Thus, the root node always has a probability of 1.00 because it summarizes all of the instances.

- A list of attribute/value pairs annotated with the probability that an instance that is described by that node has that particular attribute/value pair. For continuous attributes, the mean and standard deviation are kept instead of this probability.

A node that summarizes two of the bitstrings in the 12-bit representation described above (Figure 3-2) is shown in Table 3.4.

When incorporating a new instance into a concept hierarchy, COBWEB either merges it into an already existing node or creates a new node just for that instance. In addition, COBWEB can merge two nodes or split a node. Two nodes are merged when they become too similar, and a node is split into two nodes when a node becomes too general.

COBWEB chooses among these four node operations by using an evaluation function called category utility [Gluck and Corter, 1985]. The option which has the highest category utility is performed. This corresponds to finding a set of nodes which maximizes the difference between the probability that an instance has a certain at-

$P(C) = 1.00$		$P(V\|C)$
First	0	0.5
	1	0.5
Second	0	1.0
	1	0.0
Third	0	1.0
	1	0.0
Fourth	0	0.0
	1	1.0
Fifth	0	0.0
	1	1.0
Sixth	0	1.0
	1	0.0
Seventh	0	0.5
	1	0.5
Eighth	0	1.0
	1	0.0
Ninth	0	0.0
	1	1.0
Tenth	0	0.5
	1	0.5
Eleventh	0	0.0
	1	1.0
Twelfth	0	0.0
	1	1.0

Table 3.4: A node that summarizes two instances. The two instances are "0 0 0 1 1 0 0 0 1 1 1 1" and "1 0 0 1 1 0 1 0 1 0 1 1" and are taken from the 12-bit representation shown above. The first instance is the bitstring for trytophan and the second instance is the bitstring for tyrosine. These two instances are grouped together by the clustering algorithm and so it is not surprising that nine of the twelve attributes are identical. The attribute names refer to the bit positions. So, for example, the eighth position of both of the instances is "0" so the probability that the value of the eighth attribute is "0" given that the instance is described by this node is 1.0.

tribute/value pair given its class and the probability that an instance has a certain attribute/value pair given no class information. The first value is stored in the node. The second value is taken to be the probability that the instances summarized by the parent node have that attribute/value pair. Thus, category utility favors forming nodes that are different from their parents.

Decision tree system

A public domain version of Quinlan's C4 decision tree classifier was used in this thesis. This decision tree system takes as input the clustering provided by COBWEB and the database of biochemical properties shown in Table 3.1 and produces a decision tree that explains the clustering in terms of the biochemical properties. This decision tree is used to classify instances. Each node of the tree contains a test on an attribute. The branches that exit from a node correspond to the outcomes of the test. The leaves of the tree contain classes.

The tests at a node depend on whether the attribute is continuous or discrete. If the attribute is continuous then the test is of the form $value > N$ where N is a constant. The two branches that exit these nodes correspond to the test being true or false. If the attribute is discrete, then there is a branch for each value that the attribute can have.

The C4 algorithm creates a decision tree by cycling through all of the possible tests and choosing the one that maximizes an information theoretic metric. Each test splits the set of instances into subsets. The procedure is applied recursively on these subsets until all of the instances in a subset belong to one class. At that point a leaf node annotated with that class is created, and no further subdivision is performed.

3.3 Results

After the genetic algorithm and neural network parameters discussed above have been set, the primary parameter that controls the prediction accuracy of the representations is the number of bits used to encode each amino acid. The first part of this

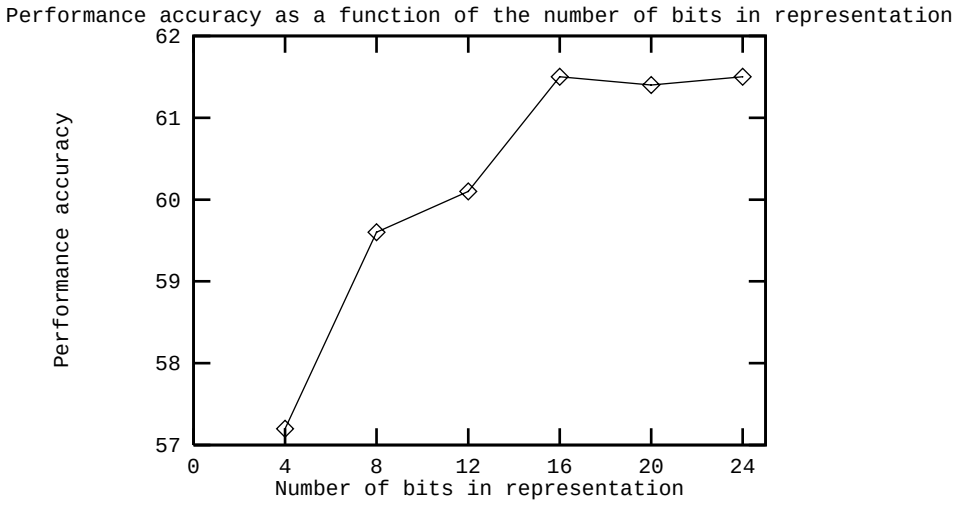

Figure 3-12: Performance accuracy as a function of the number of bits used to represent an amino acid. The performance accuracy is the percentage of secondary structure instances that the neural network predicts correctly. So, for example, the neural net trained using the best 12 bit representation found to date correctly identifies 60.1% of the secondary structure. Since the graph peaks at 16 bits, most of my efforts have been concentrated on generating good 16-bit representations.

section describes a set of experiments that led us to further explore 16-bit representations. The second part describes the best 16-bit representation that we have found.

3.3.1 How many bits should a representation have?

There are two opposing forces that determine the number of bits that should be used in a representation.

On the one hand, the number of bits should be high because this increases the expressive power of the representation. On the other hand, the number of bits should be low so that the space can be searched thoroughly. We have searched the space of representations that have 4, 8, 12, 16, 20, and 24 bits per amino acid. The results are summarized in Figure 3-12. The graph peaks at 16 bits, and therefore we have concentrated most of our efforts on generating good 16-bit representations.

```
1 0 0 1 0 1 1 1 1 0 1 1 1 0 0 0    Alanine
1 0 0 1 1 0 0 1 0 1 0 0 1 1 0 0    Cysteine
1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0    Aspartic acid
0 0 1 1 1 1 1 0 1 1 0 1 1 1 1 0    Glutamic acid
0 0 1 0 1 0 0 0 0 0 0 1 1 0 1 1    Phenylalanine
1 0 0 1 0 0 1 0 1 1 0 0 1 1 0 1    Glycine
1 1 1 0 1 0 0 0 1 0 0 0 0 0 1 1    Histidine
0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1    Isoleucine
1 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0    Lysine
1 0 1 1 1 1 0 1 0 1 1 1 1 0 0 1    Leucine
1 1 1 0 1 1 0 1 0 0 1 0 0 0 1 0    Methionine
1 0 0 1 1 1 1 0 1 1 0 0 1 0 0 1    Asparagine
0 1 0 1 1 0 0 0 1 0 0 0 0 1 0 0    Proline
0 0 1 0 0 1 1 0 0 1 0 0 1 1 1 0    Glutamine
0 1 0 1 0 0 0 1 0 0 1 1 0 0 1 1    Arginine
0 1 0 0 1 1 1 1 0 1 1 1 1 0 1 1    Serine
1 1 0 0 0 1 0 0 1 0 0 0 1 1 0 1    Threonine
1 1 0 1 1 1 0 0 1 1 1 0 0 1 1 1    Valine
0 0 0 0 1 1 0 0 1 1 1 1 0 1 1 0    Tryptophan
1 0 1 1 0 1 1 0 0 1 0 1 0 1 0 1    Tyrosine
```

Figure 3-13: The best amino acid representation found to date. Each row is the representation for an amino acid. The entire 16x20 matrix is the amino acid representation. The genetic algorithm searches over the space of these representations to find the best one. Only the twenty bitstrings that correspond to the twenty amino acids are shown.

3.3.2 The best 16-bit representation

The best 16 bit amino acid representation we have found is shown in Figure 3-13 and the results of clustering this representation are shown in Figure 3-14.

The decision tree system uses a table of biochemical properties of amino acids, shown in Table 3.1, to explain the clustering of the amino acids. The decision tree is shown in Figure 3-15. The decision tree system uses the three attributes thought to be the most important for tertiary structure prediction (bulk, hydrophobicity, and charge (pI)) to explain the clustering.

```
Concept hierarchy is:
N2
   N22
      TRYPTOPHAN-W
      TYROSINE-Y
      GLUTAMINE-Q
      CYSTEINE-C
      LEUCINE-L
      GLUTAMIC-ACID-E
   N20
      VALINE-V
      PROLINE-P
      LYSINE-K
   N18
      THREONINE-T
      ASPARAGINE-N
      GLYCINE-G
   N24
      SERINE-S
      ARGININE-R
      METHIONINE-M
   N11
      ISOLEUCINE-I
      HISTIDINE-H
      PHENYLALANINE-F
   N4
      ASPARTIC-ACID-D
      ALANINE-A
```

Figure 3-14: Clustering of amino acids based on the best 16 bit amino acid representation. The amino acids have been grouped into six clusters. The decision tree system explains these clusters by finding the biochemical properties that are shared by the amino acids in each cluster. The amino acid representation contains bitstrings for twenty four elements, but the clustering is done only over the twenty amino acids.

```
Decision Tree:

Bul <= -0.34
|   Hyd = yes
|   Hyd = no
|   |   pI <= 5.02
|   |   pI > 5.02
Bul > -0.34
|   pI > 7.58
|   pI <= 7.58
|   |   Hyd2 <= 1.14
|   |   Hyd2 > 1.14
```

Figure 3-15: Decision tree. The decision tree shows that bulk, hydrophobicity, and charge(pI) are the biochemical features captured by the 16 bit amino acid representation. The combinations of these attributes produced by the decision tree system can be viewed as pseudo-attributes derived from the amino acid representation and the database of amino acid biochemical properties.

3.3.3 Understanding the 16-bit representation

The 16-bit representation shown in Figure 3-13 and the clustering formed from this representation (Figure 3-14) and the decision tree formed from this clustering (Figure 3-15) bear closer examination.

Pair-wise Hamming distances of the amino acids in the clusters

COBWEB creates the clustering shown in Figure 3-14 by maximizing the intra-group similarity in Hamming distance space and minimizing the inter-group similarity. Figure 3.5 shows the intra-group Hamming distances for each pair of amino acid representations in the six clusters. The average pair-wise Hamming distances for the six clusters (in the order given in Figure 3.5) is: 7.3, 6, 4.7, 6.7, 4, 3.

The expected Hamming distance between two random bitstrings of length 16 is 8. Thus, all six clusters have average Hamming distances that are less than this distance, as expected. In addition, the largest cluster has the highest average Hamming distance and the smallest cluster has the lowest average Hamming distance, also as expected.

In the first cluster, the lowest Hamming distance is between glutamine and glu-

tamic acid. Glutamine is the uncharged derivative of glutamic acid. However, not all of the low Hamming distances can be explained so easily. In the third cluster, asparagine and glycine have the lowest pair-wise Hamming distance, but there appears to be no obvious relationship between them. It is for these more complicated cases that the interdependencies highlighted by the decision tree in Figure 3-15 are required. Both glycine and asparagine have bulk values less than or equal to -.34, both are not hydrophobic, and both have pI values greater than 5.02.

Perturbing the database

If one of the attributes that the decision tree uses to classify the clusters is eliminated from the database, then the decision tree will be forced to use other attributes. The way in which the tree changes illuminates how the attributes interact.

Figure 3-16 shows how the decision tree compensates for the elimination of the discrete hydrophobicity attribute. The tree is quite similar to the original tree. In particular, it does not use any additional attributes. Figure 3-17 shows the decision tree after the continuous hydrophobicity attribute has been deleted from the database. Again, the tree uses only bulk, hydrophobicity, and charge (pI). Figure 3-18 shows the decision tree after the charge (pI) attribute has been removed from the database. The top level test remains "Bul $\leq$ -0.34" and the only two attributes used are bulk and hydrophobicity. Finally, Figure 3-19 shows the decision tree after the bulk attribute has been removed from the database. This decision tree uses only the charge (pI) and continuous hydrophicity attributes. Thus, none of the decision trees ever use any other attributes in addition to the ones used by the original decision tree.

These results suggest that the tree is quite robust with respect to changes in the database. Bulk, hydrophobicity, and charge (pI) and their interdependencies do in fact seem to give a good explanation of the clustering shown in Figure 3-14.

These decision trees can also be used to understand how attributes compensate for one another. The second branch ("Bul $>$ -0.34") of Figure 3-16, the decision tree that does not have access to the discrete hydrophobicity attribute, is identical to the second branch of Figure 3-15. The first branch ("Bul $\leq$ -0.34") contains some

(A)

	W	Y	Q	C	L	E
W	-	9	7	9	9	5
Y	-	-	6	8	6	6
Q	-	-	-	8	10	4
C	-	-	-	-	6	8
L	-	-	-	-	-	8
E	-	-	-	-	-	-

(B)

	V	P	K
V	-	6	7
P	-	-	5
K	-	-	-

(C)

	T	N	G
T	-	6	5
N	-	-	3
G	-	-	-

(D)

	S	R	M
S	-	6	7
R	-	-	7
M	-	-	-

(E)

	I	H	F
I	-	4	3
H	-	-	5
F	-	-	-

(F)

	D	A
D	-	3
A	-	-

Table 3.5: Hamming distances between pairs of amino acids in the same clustering. (A) First cluster: W=tryptophan, Y=tyrosine, Q=glutamine, C=cysteine, L=leucine, E=glutamic acid. (B) Second cluster: V=valine, P=proline, K=lysine. (C) Third cluster: T=threonine, N=asparagine, G=glycine. (D) Fourth cluster: S=serine, R=arginine, M=methionine. (E) Fifth cluster: I=isoleucine, H=histidine, F=phenylalanine. (F) Sixth cluster: D=aspartic acid, A=alanine.

```
Bul <= -0.34
|   pI <= 5.02
|   pI > 5.02
|   |   Hyd2 <= -0.03
|   |   Hyd2 > -0.03
Bul > -0.34
|   pI > 7.58
|   pI <= 7.58
|   |   Hyd2 <= 1.14
|   |   Hyd2 > 1.14
```

Figure 3-16: Decision tree after the discrete hydrophobicity attribute has been eliminated from the database. The structure of the tree is very similar to the one in Figure 3-15. The top level test is identical and the decision tree uses bulk, hydrophobicity, and charge (pI).

```
Bul <= 0.16
|   Hyd = yes
|   Hyd = no
|   |   pI <= 5.02
|   |   pI > 5.02
Bul > 0.16
|   pI > 7.58
|   pI <= 7.58
|   |   pI <= 5.88
|   |   pI > 5.88
```

Figure 3-17: Decision tree after the continuous hydrophobicity attribute has been eliminated from the database. The top level test still uses the bulk attribute, although the cutoff point has changed from -0.34 to 0.16. In addition bulk, hydrophobicity, and charge (pI) are still the only attributes used.

```
Bul <= -0.34
|   Hyd = yes
|   Hyd = no
Bul > -0.34
|   Hyd2 <= -1.07
|   Hyd2 > -1.07
|   |   Hyd2 <= 1.14
|   |   Hyd2 > 1.14
```

Figure 3-18: Decision tree after the charge (pI) attribute has been eliminated from the database. The decision tree uses only the bulk and hydrophobicity attributes.

```
pI <= 5.88
pI > 5.88
|   Hyd2 <= -0.72
|   Hyd2 > -0.72
|   |   pI <= 5.97
|   |   pI > 5.97
```

Figure 3-19: Decision tree after the bulk attribute has been eliminated from the database.

interesting differences. In Figure 3-15 the second test in the first branch of the tree uses the discrete hydrophobicity attribute ("Hyd = yes") and the third test uses the charge (pI) attribute ("pI $\leq$ 5.02"). The decision tree in Figure 3-16 does not have access to the discrete hydrophobicity attribute so it raises the test on the charge (pI) attribute to the second level ("pI $\leq$ 5.02") and uses a test on the continuous hydrophobicity attribute at the third level ("Hyd2 $\leq$ -0.03").

These attribute substitutions should split the amino acids in similar ways. As an illustration, compare Figure 3-15 and Figure 3-18. The first branch ("Bul $\leq$ -0.34") of Figure 3-18, the decision tree that does not have access to the charge (pI) attribute, is identical to the first branch of Figure 3-15 with the exception that it does not use the charge (pI) attribute. The second branch ("Bul $>$ -0.34") is also similar. The test of the continuous hydrophobic attribute is the same ("Hyd2 $\leq$ 1.14"), but the pI test (pI $>$ 7.58) has been replaced by another test on the continuous hydrophobic attribute ("Hyd2 $\leq$ -1.07"). Thus, these two decision trees indicate that the set of instances that have a bulk value greater than -0.34 and a hydrophobicity value less than or equal to -1.07 is similar to the set of instances that have a bulk value greater than -0.34 and a pI value greater than 7.58. In fact, the two sets are identical.

Using the decision tree system alone

The decision tree system can be used to learn the mapping from primary to secondary structure in much the same way that the neural network is used in the current system. In order to allow direct comparison to the decision tree shown in Figure 3-15, the amino acids are described using the biochemical properties in Figure 3.1, instead of using the bitstring representation in Figure 3-13. As with the neural networks, a window size of 13 is used. This experiment has been tried using neural networks [Qian and Sejnowski, 1988], but to the best of my knowledge it has not been tried with decision trees.

Unfortunately, the results are not interesting. The unpruned decision tree has a depth greater than one hundred and has over 2500 nodes. The pruned decision tree has just one leaf node that is annotated with coil, the most prevalent form of

secondary structure.

3.4 Control experiments

In order to further understand how the first part of this system generates good amino acid representations we have run several control experiments.

These experiments are designed to answer, in part, the following questions:

- Does the genetic algorithm converge to representations that can be explained by similar decision trees?
- What effect does changing the number of hidden units have on the prediction accuracy of the neural network?
- What effect does changing the inertia have on the prediction accuracy of the neural network?
- What effect do the initial weights of the neural net have on prediction accuracy?

3.4.1 Does the genetic algorithm converge to representations that can be explained by similar decision trees?

The best representation found so far is the 16-bit representation that is described above. This 16-bit representation is explained by a decision tree that uses bulk, hydrophobicity, and charge(pI). Ideally, the genetic algorithm should regularly converge to representations that can be explained by these three attributes. If this is not the case, then the 16-bit representation described above risks being dismissed as a fortuitous accident.

Fortunately, the genetic algorithm does converge to representations that are similar. The 12-bit representation shown above (Figure 3-2) leads to a decision tree that uses four attributes, including the three used to explain the 16-bit representation. In addition, the second best 16-bit representation is explained by a decision tree that uses

three attributes: hydrophobicity, charge(pI), and aromaticity. Thus, hydrophobicity and charge(pI) are used in the decision trees for all three representations.

3.4.2 What effect does changing the number of hidden units have?

Neural networks with three hidden units were compared to perceptrons (neural networks with no hidden units). Other researchers have found that increasing the number of hidden units does not improve prediction accuracy and often decreases it [Qian and Sejnowski, 1988, Holley and Karplus, 1989]. My results are in agreement with these findings. Neural networks with three hidden units were trained using the orthogonal amino acid representation and were found to give prediction accuracies 1% lower than perceptrons trained with the same representation. This difference is considered to be significant by researchers in the field: Holley and Karplus [Holley and Karplus, 1989], for example, choose a window of size 17 instead of one of size 15 because the neural network with the window size of 17 has a performance accuracy that is .3% higher.

In addition to having a higher performance accuracy, less time is required to train perceptrons since they have fewer weights. Also, the neural networks with hidden units suffer from overfitting, while the perceptrons appear to be less affected by this problem. For these reasons, we used neural networks with no hidden units (perceptrons) throughout this work.

3.4.3 What effect does changing the learning rate have?

Figure 3-20 shows how performance accuracy changes as a function of inertia for a neural network trained for 200 epochs with a learning rate of 0.00001 and three hidden units. The figure shows that the performance accuracy changes drastically as a function of this parameter. As mentioned immediately above perceptrons seem to be less sensitive to changes in these parameters and, partly for this reason, they were chosen for this system.

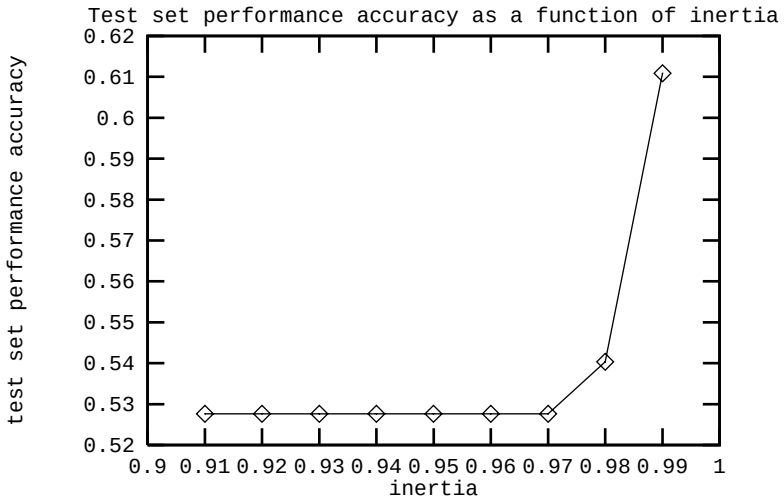

Figure 3-20: Performance accuracy of a neural network as a function of the inertia for a neural network with three hidden units. The performance is extremely brittle with respect to the inertia: a 1% change from 0.98 to 0.99 improved the performance accuracy by more than 7%. The neural networks with inertias between 0.91 and 0.97 always predict coil.

Figure 3-21 shows that the performance accuracy of the perceptron is much more robust with respect to inertia.

3.4.4 What effect does changing the initial neural net weights have?

The initial weights of the neural network are randomly set to small values. To investigate whether or not the initial weights affect prediction accuracy, we trained a neural network on a single representation ten times, each time with a different set of initial weights. The results are shown in Table 3.6. The table shows that the effect of the initial weights on the final prediction accuracy is small.

3.5 Extending the learning algorithm

As explained above the components of the system can be modified in various ways. Here we describe one such extension.

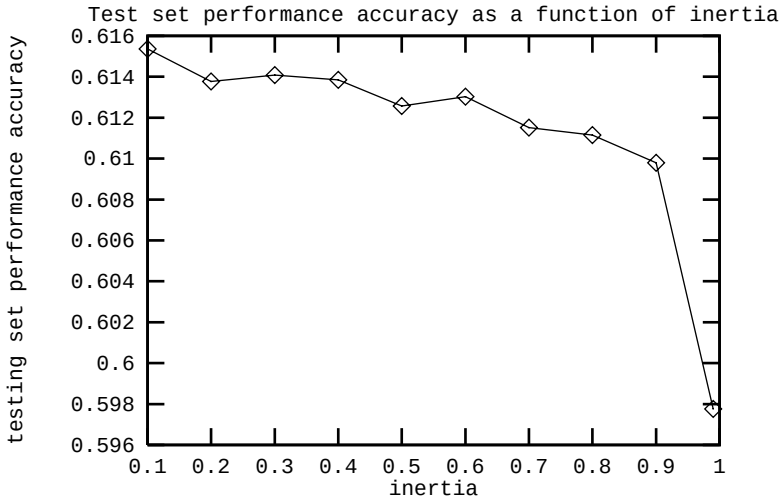

Figure 3-21: Test set performance accuracy as a function of inertia for a perceptron. The performance accuracy of the perceptron is robust over a wide range of inertias, unlike the performance accuracy of the neural network with three hidden units.

Training set prediction accuracy	Testing set prediction accuracy
0.665702	0.599564
0.665702	0.599564
0.666006	0.599865
0.666311	0.598436
0.666616	0.599865
0.666616	0.599865
0.669359	0.602722
0.669359	0.602722
0.669359	0.602722
0.669359	0.602722

Table 3.6: Results of training a neural net with different initial weights. This table shows that the effect of the initial weights on the final prediction accuracy is small. The range of the training set prediction accuracies is less than .004 and the range of the test set prediction accuracies is less than .005.

```
0.759225 0.128525 0.045692 b
0.854885 0.090735 0.047074 b
0.663669 0.121686 0.135690 b
0.229535 0.416797 0.312370 c
0.198064 0.295840 0.394201 c
0.299105 0.299105 0.302391 c
0.383064 0.094678 0.459076 c
0.649579 0.061424 0.305697 c
0.738669 0.069038 0.221351 c
0.700895 0.079782 0.325950 a
0.616936 0.107437 0.243642 a
0.564098 0.116764 0.282988 a
```

Figure 3-22: Input to the memory-based learning algorithm. A memory-based learning algorithm was added as a post-processor to the neural network. The neural network produces three numbers, ranging from 0 to 1, corresponding to the three secondary structure prediction classes (alpha helix, beta sheet, and coil). The memory-based learning algorithm takes as input these three numbers and predicts the secondary structure.

Maron and Moore [Maron and Moore, 1993] discuss a technique for selecting a good memory-based learning algorithm from a set of learning algorithms. We added this technique as a post-processor to the predictions produced by the neural network. This refinement improved the secondary structure prediction by 1.8% $\pm$ 0.9%. The idea behind this post-processing is similar to one employed by Zhang *et al.*[Zhang et al., 1992] who use a neural network to decide amongst the predictions made by three experts (a neural network, a memory-based system, and a statistical algorithm).

A subset of the data used by this system is shown in Figure 3-22. The first three columns are the outputs of the neural network and the fourth column is the class (a for alpha helix, b for beta sheet, and c for coil).

Maron [Maron, 1993] describes a technique that partitions instances into easily learnable regions and shows that partitioned algorithms outperform non-partitioned algorithms by 10-30% on a set of problems taken from the UC-Irvine machine learning depository. This method may further improve the accuracy of the system.

Chapter 4

Summary and Future work

This thesis has described a system that synthesizes regularity exposing attributes in large protein databases. In addition, it shows that given certain assumptions, learning a mapping from unannotated primary structure sequences to secondary structure requires a prohibitive number of instances.

The best representation discovered by the system is 33% shorter than the traditional orthogonal representation, and yet, a neural network trained with that representation achieves a performance accuracy that is equal to that of a neural network trained with the traditional representation. If the output of the neural network is processed by a memory-based reasoning system, then the performance accuracy is 1.8% higher than that of a neural network trained with the traditional representation.

The system that produces this representation is divided into four subsystems:

- A search algorithm. In this thesis, a genetic algorithm is used.
- An algorithm that quantitatively evaluates the quality of a representation. Here the performance accuracy of a neural network trained with the representation is this measure of quality.
- A clustering algorithm. The clustering algorithm in this thesis uses the bitstring representation of each amino acid to group the amino acids whose bitstrings are similar in Hamming distance space.

- A learning algorithm that uses a database of amino acid biochemical properties to predict the cluster of an amino acid.

The first two subsystems are used in tandem to produce a representation that facilitates secondary structure prediction. The second two subsystems explain this representation in terms of amino acid biochemical properties. The best representation produced by this system can be explained in terms of bulk, hydrophobicity, and charge(pI), three of the attributes thought to be most important for predicting tertiary structure.

There are several ways in which this thesis might be extended. These extensions fall into two categories: fine-tuning of the current system and application of the methods embodied by the system to other problems.

There are at least three ways to fine-tune the system:

- First, the representations found by the genetic algorithm might be fine-tuned for superior performance. Although, as shown in Chapter 3, there is some evidence that the genetic algorithm is converging, the number of generations is too low to do a fine-grained search around the final representation.

- Second, as mentioned in the text, other algorithms can be substituted for one of the four learning algorithms used in the system as long as they have the same function. In particular, the neural network, which is currently the slowest part of the system, might be fruitfully replaced by a faster learning algorithm.

- Third, the clustering algorithm could be adjusted to identify the bit positions that are most important for the clustering. This might allow the bits to be given a biochemical interpretation. For example, bits one through three might be found to encode the hydrophobicity of the amino acid.

Finally, the ideas embodied by the system might be applied to other problems in computational biology. For example, many of the current approaches to the tertiary structure prediction problem explicitly represent the coordinates of the atoms in the

protein. A system like the one described in this thesis might be able to find higher-order features in proteins and therefore simplify the problem.

Appendix A

Appendix

This appendix contains operational details about the software used in this thesis. Examples of the files manipulated by these programs are provided and explained for the four main subsystems used in the thesis. Information about how to acquire the software is given in Table A.1. The appendix is divided into four parts, one for each of the four subsystems.

A.1 Genetic algorithm: GENEsYs

The GENEsYs genetic algorithm package allows the user to specify a wide range of genetic algorithms through command line options. The five parameters discussed in Chapter 3 (crossover rate, mutation rate, number of individuals, number of generations, and the selection procedure) can all be set. Furthermore, several other parameters, including the number of experiments to perform, the number of bits per individual, and the format of report files, can also be set using command line options.

Name	Description	Language	Source
GAucsd	genetic algorithm	C	cs.ucsd.edu (ftp)
GENEsYs	genetic algorithm	C	lumpi.informatik.uni-dortmund.de (ftp)
COBWEB	clustering algorithm	Lisp	cobweb@ptolemy.arc.nasa.gov (request)
ASPIRIN	neural network package	C	pt.cs.cmu.edu (ftp)
C4	decision tree system	C	quinlan@cluster.cs.su.oz.au (request)

Table A.1: Availability information for the public domain software used in this thesis.

```
ga -f 28 -P10 -U 10 -L 16 -t 100 -s 10 -g 100 -d 1 -r 9289 -o p -v 1
```

Figure A-1: Example GENEsYs call. The "-f" option specifies the function to be optimized. The "-P" and "-U" options set the number of individuals in a population to ten. The "-L" option specifies the number of bits per variable. As explained in chapter 3, there are 24 variables, one for each amino acid and four additional variables. The "-t" option sets the total number of function evaluations. The "-s", "-d", "-o", and "-v" specify the format of the final report file. The "-r" option sets the initial random number seed.

An example command line call is shown in Figure A-1.

In addition to these parameters, the user needs to provide C code that implements the function to be optimized. Figure A-2 shows the C code that implements the function that is used to search for 16-bit representations. This function takes as input an individual and returns the fitness of this individual. The function calls a neural network (described below) via a shell script.

A.2 Neural network: ASPIRIN

ASPIRIN is a language that allows a wide variety of neural network to be specified easily. In addition, the ASPIRIN package contains analysis tools that help the user understand the structure of the neural network. An ASPIRIN specification for a neural network with 312 input units, 0 hidden units, and 3 output units is shown in Figure A-3.

This neural network specification is compiled by the command "bpmake" which creates an executable program that can be called from a user procedure. In this system it is called from a shell script which is in turned called by the genetic algorithm procedure described above. The executable program saves the weights of the neural network at the end of training so that the same neural network can be used to classify instances in the cross-validation set and testing set.

```
#include "../extern.h"
double
f_28(x, n)
register char           x[];
register int            n;
{
  int i,j;
  double fitness;
  int BITS_PER_AA = 16;
  int NUM_AA = 24;

/* write individual out to individual file.  This will overwrite */
 /* previous individual.  */

  FILE *fp;
  fp = fopen("individual2", "w");
  for(i=0; i<BITS_PER_AA; i++)
    {
      for(j=0; j<NUM_AA; j++)
        fprintf(fp, "%c", x[i*NUM_AA + j]+48);
      fprintf(fp, "\n");
    }

  fclose(fp);

  /* call to quick_decode_ga.o */
  system("quick_decode_ga_16.o prot.train individual2 > ASPIRIN/prot_train_ga.data");

  /* call shell file - writes out to file answer */
  system("run_nn16.sh");

  /* read file answer */
  fp = fopen("answer", "r");
  fscanf(fp, "%lf", &fitness);
  fclose(fp);

  return(-fitness);
}
```

Figure A-2: Example GENEsYs function. This function returns the fitness of an individual.

```
#define N_OUTPUT  3
#define N_HIDDEN  0
#define N_INPUT  312

DefineBlackBox encoder
{
    OutputLayer->  Output_Layer
    InputSize->    N_INPUT
    Components->
    {
        PdpNode Output_Layer [N_OUTPUT]
            {
                InputsFrom-> $INPUTS
            }
    }
}
```

Figure A-3: ASPIRIN neural network specification. This neural network has 312 input units which are fully connected to 3 output units. The output units are sigmoidal units (specified by the key word "PdpNode"). The input units are fully connected to the output units. The N_HIDDEN declaration is not used by the program, but it serves to document the code.

```
(Alanine-A 1 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1)
(Cysteine-C 1 0 0 1 1 0 1 0 1 0 1 0 0 1 1 1)
(Aspartic-Acid-D 0 0 1 0 0 1 1 0 1 1 1 0 0 0 1 1)
(Glutamic-acid-E 0 0 0 1 1 0 1 1 0 1 1 1 1 1 0 0)
(Phenylalanine-F 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 1)
(Glycine-G 0 0 0 0 0 1 0 0 1 1 0 1 1 1 1 0)
(Histidine-H 1 0 0 0 1 1 0 1 1 1 0 0 0 0 1 0)
(Isoleucine-I 0 1 1 0 1 0 1 0 1 0 0 0 0 1 1 1)
(Lysine-K 1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0)
(Leucine-L 1 1 1 1 0 0 1 1 0 1 0 1 0 1 1 0)
(Methionine-M 1 1 1 1 0 0 1 0 0 1 1 0 1 1 1 0)
(Asparagine-N 1 1 1 0 0 0 0 0 1 0 1 1 1 1 1 1)
(Proline-P 0 1 1 1 1 1 1 0 0 1 1 0 1 0 1 1)
(Glutamine-Q 0 1 1 0 0 0 0 1 0 0 0 0 0 1 1 1)
(Arginine-R 1 1 0 0 0 1 1 0 1 1 1 1 1 1 0 1)
(Serine-S 1 1 1 1 1 1 0 1 0 1 1 0 0 1 0 1)
(Threonine-T 0 0 0 1 0 0 0 0 0 1 0 0 1 1 1 0)
(Valine-V 0 0 1 1 0 0 1 1 1 0 0 1 0 0 1 0)
(Tryptophan-W 0 0 0 1 1 0 1 0 0 0 0 1 0 1 0 1)
(Tyrosine-Y 0 1 1 1 1 0 1 0 0 1 0 0 0 0 0 0)
```

Figure A-4: Example COBWEB input file. Each list is an instance. The first element of each list is a descriptor that is ignored by the system. The other elements in the list are used to cluster the instances.

A.3 Clustering algorithm: COBWEB

COBWEB is a Lisp package (unlike the rest of the systems which are all implemented in C) that clusters a set of instances. A sample input file is shown in Figure A-4. Examples of the output produced by COBWEB are given in chapter 3.

A.4 Decision tree system: C4

The C4 package takes as input a text file containing instances and a text file that contains descriptions of the attributes used to describe those instances. It produces as output a decision tree. An example file that contains instances is shown in Figure A-5 and a file that contains attribute descriptions is shown in Figure A-6. Examples of the decision trees produced by the C4 package are given in Chapter 3.

```
yes, no,  no,  yes, no,  no,  no,  no,  6.02,  0.32,  -1.44, 6 | alaninea
no,  yes, no,  no,  no,  no,  yes, no,  10.76, -1.07,  1.16, 4 | argininer
no,  no,  yes, no,  no,  no,  no,  yes, 5.41,  -.96,  -0.34, 3 | asparaginen
no,  yes, no,  no,  no,  no,  no,  yes, 2.97,  -1.07, -0.54, 6 | asparticacidd
no,  no,  yes, no,  no,  yes, no,  no,  5.02,  1.5,   -0.75, 1 | cysteinec
no,  yes, no,  no,  no,  no,  no,  yes, 3.22,  -1.03,  0.17, 1 | glutamicacide
no,  no,  yes, no,  no,  no,  no,  yes, 5.65,  -1.05,  0.22, 1 | glutamineq
no,  no,  no,  yes, no,  no,  no,  no,  5.97,  -.03,  -2.16, 3 | glycineg
no,  no,  yes, no,  no,  no,  yes, no,  7.58,   -.13,  0.52, 5 | histidineh
yes, no,  no,  yes, no,  no,  no,  no,  5.98,   1.52,  0.21, 5 | isoleucinei
yes, no,  no,  yes, no,  no,  no,  no,  5.98,  1.14,   0.25, 1 | leucinel
no,  yes, no,  no,  no,  no,  yes, no,  9.74,  -1.76,  0.68, 2 | lysinek
yes, no,  no,  no,  no,  yes, no,  no,  5.75,   1,     0.44, 4 | methioninem
yes, no,  no,  no,  yes, no,  no,  no,  5.98,   1.16,  1.09, 5 | phenylalaninef
yes, no,  no,  no,  no,  no,  no,  no,  6.1,   -.72,  -0.71, 2 | prolinep
no,  no,  yes, no,  no,  no,  no,  no,  5.68,  -.46,  -1.21, 4 | serines
no,  no,  yes, no,  no,  no,  no,  no,  6.53,  -.36,  -0.67, 3 | threoninet
no,  no,  yes, no,  yes, no,  no,  no,  5.88,  0.67,   2.08, 1 | tryptophanw
no,  no,  yes, no,  yes, no,  no,  no,  5.65,  -.07,   1.34, 1 | tyrosiney
yes, no,  no,  yes, no,  no,  no,  no,  5.97,  1.38,  -0.34, 2 | valinev
```

Figure A-5: Example C4 instance file. This file contains twenty instances that correspond to the twenty amino acids. The first eleven columns are attributes and the twelfth column is the cluster that the instance belongs to. C4 generates a decision tree that uses the attributes to predict the cluster. Each instance is followed by the name of the amino acid that it corresponds to ("|" is the comment character).

```
|  Hydrophobic, charged, polar from Intro. to protein
|  structure.  Aliphatic, aromatic, sulfur, basic, acidic from Stryer
|  Biochemistry.  pi from Mahler Biological Chemistry


1, 2, 3, 4, 5, 6, 7 | Classes
|name:       tryptophanw, tyrosiney, glutamineq, cysteinec, leucinel,
|            glutamicacide, valinev, prolinep, lysinek, threoninet,
|            asparaginen, glycineg, serines, argininer, methioninem,
|            isoleucinei, histidineh, phenylalaninef, asparticacidd, alaninea
hydrophobic: yes, no
charged    : yes, no
polar      : yes, no
aliphatic  : yes, no
aromatic   : yes, no
sulfur     : yes, no
basic      : yes, no
acidic     : yes, no
pi         : continuous  | from Mahler Biological Chemistry
hydrophobic2: continuous | from Kidera, Scheraga (prot. chem. '85 - type 3)
bulk       : continuous  | from Kidera, Scheraga (prot. chem. '85 - type 3)
```

Figure A-6: Example C4 attributes file. This file indicates whether each attribute is discrete or continuous. If the attribute is discrete then it is annotated with the values that it can have. The first uncommented line of the file ("|" is the comment character) is a list of the classes that the instances can belong to. In this system the classes correspond to the clusters produced by the clustering algorithm.

Bibliography

[Aha et al., 1991] Aha, D., Kibler, D., and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 6:37–66.

[Amsterdam, 1988] Amsterdam, J. (1988). Extending the Valiant learning model. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 381–394.

[Argos, 1990] Argos, P. (1990). Analysis of sequence-similar pentapeptides in unrelated protein tertiary structures: Strategies for protein folding and a guide for site-directed mutagenesis. *Journal of Molecular Biology*, 197:331–348.

[Bäck and Hoffmeister, 1992] Bäck, T. and Hoffmeister, F. (1992). A user's guide to *GENEsYs 1.0*. Software package documentation.

[Baum and Haussler, 1989] Baum, E. and Haussler, D. (1989). What size net gives valid generalization? In Touretzky, D., editor, *Advances in Neural Information Processing Systems I*, pages 81–90. Morgan Kaufmann, San Mateo, CA.

[Blum and Rivest, 1992] Blum, A. and Rivest, R. (1992). Training a 3-node neural network is NP-complete. *Neural Networks*, 5:117–127.

[Blumer et al., 1986] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. (1986). Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 273–282.

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

[Breiman et al., 1984] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth, Belmont, CA.

[Caldwell and Johnston, 1991] Caldwell, C. and Johnston, V. (1991). Tracking a criminal suspect through 'face-space' with a genetic algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 416–421.

[Cheeseman et al., 1988] Cheeseman, P., Kelly, J., Self, M., Stutz, J., W., T., and Freeman, D. (1988). Autoclass: A Bayesian classification system. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 54–64.

[Chou and Fasman, 1974] Chou, P. and Fasman, G. (1974). Prediction of protein conformation. *Biochemistry*, 13:222–244.

[de la Maza and Tidor, 1993] de la Maza, M. and Tidor, B. (1993). An analysis of selection procedures with particular attention paid to proportional and Boltzmann selection. In Forrest, S., editor, *Genetic Algorithms: Proceedings of the Fifth International Conference (GA93)*, San Mateo, CA. Morgan Kaufmann.

[Ehrenfeucht et al., 1988] Ehrenfeucht, A., Haussler, D., Kearns, M., and Valiant, L. (1988). A general lower bound on the number of examples needed for learning. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 139–154.

[Fisher, 1987] Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172.

[Franke, 1984] Franke, R. (1984). *Theoretical Drug Design Methods*. Elsevier, New York.

[Gibrat et al., 1991] Gibrat, J.-F., Robson, B., and Garnier, G. (1991). Influence of the local amino acid sequence upon the zones of the torsional angles φ and ψ adopted by residues in proteins. *Biochemistry*, 30:1578–1586.

[Gluck and Corter, 1985] Gluck, M. and Corter, J. (1985). Information, uncertainty and the utility of categories. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 283–287.

[Goldberg, 1989] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA.

[Grefenstette et al., 1985] Grefenstette, J., Gopal, R., Rosmaita, B., and van Gucht, D. (1985). Genetic algorithms for the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 160–165.

[Haussler, 1989] Haussler, D. (1989). Generalizing the PAC model for neural net and other learning applications. Technical report, University of California, Santa Cruz, CA.

[Holland, 1975] Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor.

[Holley and Karplus, 1989] Holley, L. and Karplus, M. (1989). Protein secondary structure prediction with a neural network. *Proceedings of the National Academy of Science*, 86:152–156.

[Jones, 1992] Jones, M. (1992). Using recurrent networks for dimensionality reduction. AI Tech Report 1396, Massachusetts Institute of Technology.

[Kabsch and Sander, 1984] Kabsch, W. and Sander, C. (1984). On the use of sequence homologies to predict protein structure: Identical pentapeptides have completely different conformations. *Proceedings of the National Academy of Science*, 81:1075–1078.

[Kearns and Vazirani, 1992] Kearns, M. and Vazirani, U. (1992). The PAC model and Occam's Razor. Preliminary draft of forthcoming book.

[Kidera et al., 1985] Kidera, A., Konishi, Y., Oka, M., Ooi, T., and Scheraga, H. (1985). Statistical analysis of the physical properties of the 20 naturally occurring amino acids. *Journal of Protein Chemistry*, 4(1):23–55.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, Jr., C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

[Kneller et al., 1990] Kneller, D., Cohen, F., and Langridge, R. (1990). Improvements in protein secondary structure prediction by an enhanced neural network. *Journal of Molecular Biology*, 214:171–182.

[Lander et al., 1991] Lander, E., Langridge, R., and Saccocio, D. (1991). Mapping and interpreting biological information. *Communications of the ACM*, 34(11):33–39.

[Langley, 1980] Langley, P. (1980). Descriptive discovery processes: Experiments in Baconian science. Rep. No. CS-80-121, Carnegie-Mellon University.

[Leighton, 1992] Leighton, R. (1992). The Aspirin/MIGRAINES neural network software. Software package documentation.

[Lenat, 1976] Lenat, D. (1976). AM: An artificial approach to discovery in mathematics as heuristic search. Rep. No. STAN-CS-76-570, Stanford University.

[Lim, 1974] Lim, V. (1974). Algorithms for prediction of α-helical and β-structural regions in globular proteins. *Journal of Molecular Biology*, 88:873–894.

[Mahler, 1971] Mahler, H. R. (1971). *Biological Chemistry*. Harper & Row, New York, second edition.

[Maron, 1993] Maron, O. (1993). Efficient use of erro¡r for creating piecewise learnable partitions. Unpublished draft.

[Maron and Moore, 1993] Maron, O. and Moore, A. (1993). Fast model selection using Hoeffding races. In *Advances in Neural Information Processing Systems*, volume 6.

[Martin et al., 1989] Martin, Y., Kutter, E., and Austel, V., editors (1989). *Modern Drug Research.* Marcel Dekker, Inc., New York.

[McKusick and Thompson, 1990] McKusick, K. and Thompson, K. (1990). Cobweb/3: A portable implementation. Technical Report FIA-90-6-18-2, NASA Ames Research Center, Artificial Intelligence Research Branch, Moffet Field, CA.

[Michalewicz, 1992] Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin.

[Michalski and Stepp, 1992] Michalski, R. and Stepp, R. (1992). Clustering. In Shapiro, S., editor, *Encyclopedia of Artificial Intelligence (Vol. 1)*, pages 168–176. John Wiley & Sons, Inc.

[Qian and Sejnowski, 1988] Qian, N. and Sejnowski, T. (1988). Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, 202:865–884.

[Quinlan, 1983] Quinlan, J. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann.

[Quinlan, 1988] Quinlan, J. (1988). An empirical comparison of genetic and decision-tree classifiers. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 135–141.

[Quinlan, 1993] Quinlan, J. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, Inc., San Mateo, CA.

[Rivest, 1987] Rivest, R. (1987). Learning decision lists. *Machine Learning*, 2(3):229–246.

[Roy and Mukhopadhyay, 1992] Roy, A. and Mukhopadhyay, S. (1992). A polynomial time algorithm for generating neural networks for classification problems. In

Proceedings of the International Joint Conference on Neural Networks, pages I:147–I:152.

[Salzberg and Cost, 1992] Salzberg, S. and Cost, S. (1992). Predicting protein secondary structure with a nearest-neighbor algorithm. *Journal of Molecular Biology*, 227:371–374.

[Schapire, 1991] Schapire, R. (1991). *The Design and Analysis of Efficient Learning Algorithms*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA.

[Schraudolph and Grefenstette, 1992] Schraudolph, N. and Grefenstette, J. (1992). A user's guide to GAucsd 1.4. Software package documentation.

[Stryer, 1988] Stryer, L. (1988). *Biochemistry*. W.H. Freeman and Company, New York, third edition.

[Sun et al., 1991] Sun, G.-Z., Chen, H.-H., Lee, Y.-C., and Giles, C. (1991). Turing equivalence of neural network¡s with second order connection weights. In *Proceedings of the International Joint Conference on Neural Networks*, pages II:357–II:362.

[Thomas and Principe, 1991] Thomas, E. and Principe, J. (1991). A simulated annealing like convergence theory for the simple genetic algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 174–181.

[Valiant, 1984] Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.

[Whitley et al., 1989] Whitley, D., Starkweather, T., and Fuquay, D. (1989). Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 133–140.

[Zhang et al., 1992] Zhang, X., Mesirov, J., and Waltz, D. (1992). Hybrid system for protein secondary structure prediction. *Journal of Molecular Biology*, 225:1049–1063.