Master's Thesis

Predicting Secondary Structure of Proteins by Linear and Dynamic Programming

David N. Perkins
dperkins@cs.arizona.edu
Department of Computer Science, University of Arizona, Tucson, AZ 85721

April 30, 2009

ABSTRACT

Proteins are sequences of amino acids that fold into secondary and tertiary structure, which plays an important role in their function. As biologists have yet to discover the rules that govern how a protein folds in nature from its underlying sequence, this thesis tries a new approach to secondary structure prediction using dynamic programming on the input protein sequence. The sequence is broken into short words, where each word has a probability of folding into the three different types of secondary structure. By combining word probabilities with an abstraction called contexts, which model a run of the same secondary structure type up to a bounded length, the optimal prediction for an entire sequence can be computed via dynamic programming. The structure probabilities for words are learned from a training set of sequences with known secondary structure using linear programming. The combined approach to prediction using linear and dynamic programming achieves high accuracy on protein sequences whose words were observed in the training set, but is far less accurate on sequences with unobserved words not seen in the training set. The challenge for future work lies in interpolating probabilities for unobserved words to achieve improved generalization.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Proteins are molecules that perform the basic functions of living creatures at the cellular level. A protein is made up of a chain of amino acids of which there are 20 different types, and their length ranges from tens to hundreds of amino acids. This chain of amino acids folds into a three dimensional structure, and this structure is very important in determining the function of a given protein. Biologists have yet to discover the rules that govern this folding process, but they know the sequence of amino acids determines the structure [9]. The three dimensional

(a) Alpha Helix (b) Beta Sheet (c) Tertiary Structure (PDB ID: 1hq8)

Figure 1.1: Secondary and Tertiary structure

structure has been discovered for a lot of proteins by using a crystallizing process, which is very difficult, time consuming, and not always possible.

There are three types of structure primary, secondary, and tertiary. Primary structure is the sequence of amino acids. Secondary structure is a localized structure that a portion of the chain of amino acids folds into, and tertiary is the full three dimensional form. There are three basic types of secondary structure, which are alpha helix, beta sheet, and loop. An alpha helix, represented by H, is a coiled structure similar to a spring, where bonds form between amino acids in adjacent coils. It takes 5 amino acids to form one full coil. A beta sheet, represented by E, is made of parallel chains where bonds form between adjacent amino acids in the different strands. A loop, represented by L, is anything not an alpha helix or a beta sheet. Figure 1.1(a) shows a representation of an alpha helix, beta sheet, and the tertiary structure of a protein. In the tertiary structure one can see the secondary structure types.

1.2 The task of secondary structure prediction

Given the amino acid sequence of a protein, the goal is to predict the correct secondary structure for each amino acid in the protein sequence. A protein and its corresponding secondary structure can be represented by two strings of equal length. The first string, S_{seq}, represents the chain of amino acids consisting of a 20 character alphabet, and the second string, S_{st}, represents each amino acid's secondary structure consisting of a 3 character alphabet, $\{E, H, L\}$. The two strings correspond such that $S_{seq}[i]$ is assigned secondary structure type $S_{st}[i]$. Below is an example of a section from the protein shown in figure 1.1:

S_{seq} :GYCGPCPNNWICHRNNCYQFFNEEKTWNQSQASCLSQNSSLLKIYSKEEQDFLKLVKSYHWMGLVQIPANGS
S_{st} : LLLLLLLLLLEEELLEEEEEEEEEELHHHHHHHHHHLLLEELLLLLHHHHHLLLLLLLLEEEEEEELLLLLL

This protein has two runs of alpha helices and several beta strands. Predictions can be

checked against proteins with known secondary structure to evaluate correctness.

1.3 Related work

There have been two main approaches to secondary structure prediction algorithms so far, neural networks and Hidden Markov Models, HMMs. Both of these methods learn values from a training set and then based on those values try to predict unknown secondary structure. The best prediction algorithms first create a profile of the target protein. A profile is a collection of similar proteins to the target protein, which involves a database search, the most common being PSI-BLAST [1]. PSIPRED [7] is the most common method used today and achieves around 80% accuracy when using profiles and around 70% accuracy when not using profiles. It uses a standard feed-forward back-propagation network architecture with a single hidden layer. A neural network is a series of layered nodes. The bottom is a layer of input nodes and the top is a layer of output nodes. Intermediate layers receive input from lower nodes and pass on computed values to higher nodes. A node can only connect to nodes in a higher layer. Back propagation is a type of learning technique developed for neural networks by Rumelhart, *et. al* [8].

The prediction techniques presented here are very different from current published approaches. Several published approaches do use short windows to predict a single position, which is common to both neural networks and HMM's, but the abstract ideas of words and contexts was not observed in any other published approaches. Constructing linear problems to learn optimal values is a common approach in many subjects, though it has not used by any published approaches. The final prediction method also predicts the structure for an entire sequence all at once.

1.4 Overview of thesis

The prediction methods presented here focus on predicting secondary structure using structure probabilities on short strings of amino acids. The prediction methods are explained in chapter 2. Chapter 3 covers computing the structure probabilities using linear programming. Interpolating probabilities on unobserved sequences by finding similar sequences is in section 3.2. Chapter 4 contains the details of the experiments run and their results. Conclusions and future research are in the last chapter, 5.

Chapter 2

Predicting The Structure Using Word Probabilities

2.1 Formulating the prediction problem

Two main ideas are used in predicting secondary structure, words and contexts. A word is a short string of amino acids with fixed length. Each word will have tendencies to fold into each of the three structures at its center. $P_s(w)$ is the probability that word w will fold into structure s at its center. For example, if $P_H(ABC) = .25$ then ABC has a .25 probability of folding into an alpha helix. Given the length of a word, WL, and the number of possible amino acids, S, the number of possible words is S^{WL}, which is exponential in the word length. If the word length is too short, there is not enough information to determine what structure the word might fold into. If the word length is too long it becomes difficult to determine probabilities for all possible words. Estimating probabilities for words is covered in chapter 3.

2.1.1 Prediction using words

The simplest prediction method computes the weighted averages of word probabilities, one for each structure, over a set window. For example at position i, for a window of length L, words

centered at positions $i - (L/2)$ to $i + (L/2)$ will be used in the averages. A weighted average is taken for each structure over the words within the window. The structure with the highest average is predicted for i, the position at the center of the window. The window is then shifted to the right by one and the process is repeated. The formula for computing the probability of structure s occurring at position i is:

$$g(i, s) = \sum_{(-L/2) \leq j \leq (L/2)} \omega_j \; P_s(w_{i+j}) \tag{2.1}$$

Note that the weights for the structures sum up to 1, $\sum_{(-L/2) \leq j \leq (L/2)} \omega_j = 1$, and the function is linear. This is important for determining the probabilities for words, which is covered in the next chapter. This method will be referred to as the word averaging prediction method throughout the rest of the thesis. Figure 2.1 has an example of computing $g(i, E)$ on a short string of amino acids.

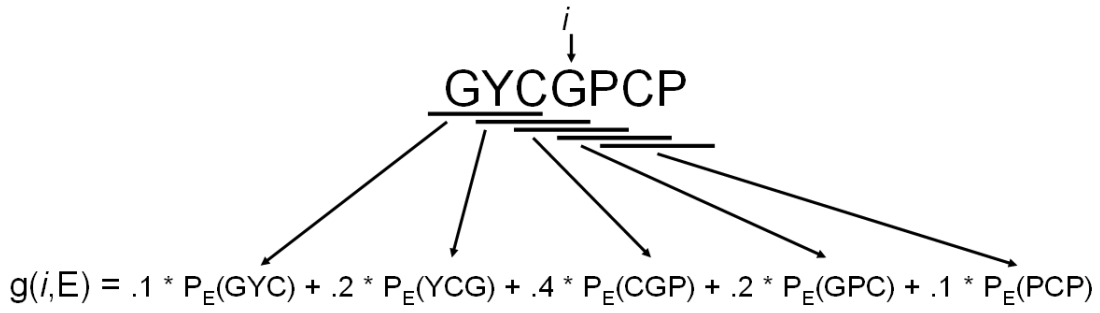

Figure 2.1: An example of computing $g(i, s)$ on a short sequence of amino acids. The word length is 3, and window length is 5, and the average weights are {.1, .2, .4, .2, .1}.

2.1.2 Prediction using contexts

The other abstraction is contexts. A context is defined by the structure at its center and how far that structure repeats to the left and to the right up to a set length. Contexts can be represented using three symbols. '+' denotes a repeat, '-' denotes a different structure, and '?' denotes any structure. For example, '?-++L+-??' represents a run of 4 loops centered on the third amino acid in the run, where the length is 4. Once the run encounters a different structure it does not matter what structures follow, which the symbol '?' represents. Given that the run can be from 0 to CL, the defined context length, the number of possible contexts is $(1 + CL)^2$. For every context, there are two probabilities. The probability of the context occurring at random, and the probability of the context occurring given one of the positions is a specific structure type. $P(c|s_j)$ equals the probability of seeing context c given structure s at position j. $P(c)$ equals the probability of the context occurring at random. The sum over all context probabilities for a single structure at position j equals 1. The probability of a given context occurring at position i in the protein sequence can be computed using context probabilities. The word probability, $P_s(w_j)$, is multiplied by the probability of seeing context c given structure s is occurring at position j. All three structure probabilities are used for every word in computing the probability ratio of one context.

$$f(i, c) = \sum_{-CL \leq j \leq CL} \omega_j \sum_{s \in \Gamma} P_s(w_{i+j}) \; P(c|s_j) \tag{2.2}$$

Γ is the set of possible structures. Figure 2.2 shows an example of computing $f(i,$'?-EEE'). The expansion of the formula can get very long, especially with larger context lengths. Prediction can be done just as before by computing the probability of every context occurring at a position and selecting the one with the highest probability. However, since there are only three different

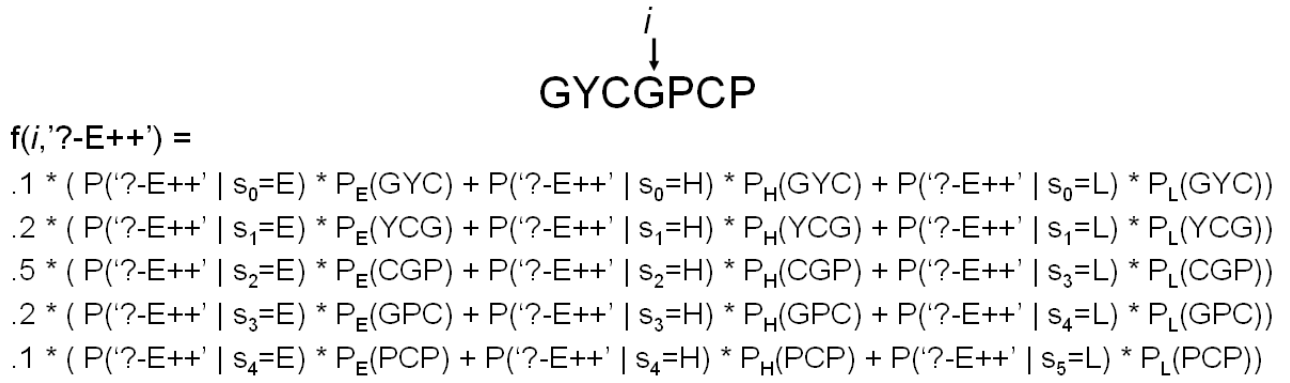

Figure 2.2: An example of computing $f(i,$'?-EEE'$)$ on a short sequence of amino acids. The word length is 3, and window length is 5, and the average weights are $\{.1, .2, .4, .2, .1\}$.

structures, all the contexts with the same structure at the center can be summed together. $F_S(i)$ stands for the sum of all $f(i,c)$ where context c has structure s at the center.

$$F_{\mathbf{S}}(i) = \sum_{c \in \mathbf{S}} f(i,c) \tag{2.3}$$

This prediction method, context averaging, fails to take into account predictions made at adjacent positions. If position i has a high probability of being context '??-+E+++-', then it should effect the probability of context '?-++E++-?' occurring at $i+1$. To accomplish this the entire sequence must be predicted together. The next equation captures this by predicting the entire sequence and maximizing the production of the prediction at each position.

$$\text{Maximize} \prod_{0 \le i < n} R(i, S[i-CL : i+CL]) \tag{2.4}$$

$S[i-CL : i+CL]$ represents the context centered at position i in the predicted structure sequence. The function R is equation 2.2 modified to be a probability ratio.

$$R(i,c) = \frac{1}{P(c)} \sum_{-CL \leq j \leq CL} \omega_j \sum_{s \in \Gamma} P_s(w_{i+j})\ P(c|s_j) \tag{2.5}$$

$\frac{1}{P(c)}$ is used to modify the probability into a ratio, because there is a bias towards structures that occur more often and in longer runs. For example, the probability for a full run of alpha helices, '++++H++++', occurring is .137 while the probability of '++++H+++-' occurring is just .023. This is understandable given alpha helices tend to occur in long runs. Consider a run of 20 helices and a context length of 4. Nine contexts with H at the center appear, eight of them just once and one context, '++++H++++', occurs twelve times. By using probability ratios, this bias is accounted for.

2.2 Computing an optimal labeling

Equation 2.4 can be solved using dynamic programming. At each position, a given context is preceded by an optimal prediction ending with a compatible context, which is limited by the context length. The problem can be solved by minimizing the sum of the negative log of $R(i, S[i - CL : i + CL])$. The recursive function $L(i, s, l, r)$ equals the optimal score of the sequence labeling up to position i, where position i has structure s, the length of the run of structure s to the left is l, and the length of the run of structure s to the right is r. s, l, and r define a specific context. Breaking contexts into these three variables helps simplify the recursion formula. $C(s, l, r)$ equals the context that s, l, and r define. Below is the recursion formula:

$$L(i,s,l,r) = \left\{ \begin{array}{cl} L(i-1,s,l-1,min\{r+1,CL\}) - \log R(i,C(s,l,r)) & \begin{array}{l} i > CL, \\ 0 < l < CL, \\ 0 \le r \le CL \end{array} \\ \\ min \left\{ \begin{array}{l} L(i-1,s,CL-1,min\{r+1,CL\}), \\ L(i-1,s,CL,min\{r+1,CL\}) \end{array} \right\} - \log R(i,C(s,l,r)) & \begin{array}{l} i > CL, \\ l = CL, \\ 0 \le r \le CL \end{array} \\ \\ min_{a \in \Gamma - \{s\}} \; min_{0 \le \tilde{l} \le CL} \{L(i-1,a,\tilde{l},0)\} - \log R(i,C(s,l,r)) & \begin{array}{l} i > CL, \\ l = 0, \\ 0 \le r \le CL \end{array} \\ \\ 0 & i = CL \end{array} \right\} \tag{2.6}$$

This recursion leads to filling in a table of size $(CL+1)^2 n$, where n is the length of the protein sequence, and then recovering the solution. Thus the dynamic algorithm takes $\Theta(CL^2 n)$ time.

2.3 Summary of methods

As a quick summary, three methods for predicting secondary structure were presented, in increasing complexity. The three methods were word averaging, context averaging, and a dynamic approach. Word averaging and context averaging are linear in the length of the protein sequence, and the dynamic approach was $\Theta(CL^2 n)$. These methods require probabilities for words and contexts, and the next chapter describes approaches to learning them. The accuracy of these prediction methods are greatly effected by the learned probabilities, so it is very important to learn good probabilities.

Chapter 3

Estimating Word and Context Probabilities

The remaining task is to find good probabilities for words and contexts. A straight forward approach is to compute empirical probabilities on a training set. Another approach is to create a linear problem based on a training set to learn probabilities. The context probabilities are computed empirically on our entire data set. The word probabilities were computed using both approaches. One of the difficulties with word probabilities is how to assign probabilities to words that were not observed in the training set. An evolutionary difference matrix was used to develop a distance metric between words, in order to relate them to one another. The evolutionary matrix contains the log odds of a single amino acid changing into another amino acid. Since, the sequence of amino acids determine structure and protein structure is highly correlated to its function, it follows evolutionary similar amino acids would have similar function and thus similar structure. The distance function $D(u, v)$ is a weighted average of the distances between pairs of amino acids in each word.

$$D(u, v) = \sum_{0 \leq i < WL} \omega_i \; d(u_i, v_i) \tag{3.1}$$

For example, the distance from ABC to DEF using a specific weighted average is $D(ABC, DEF) = (.25 * d(A, D)) + (.5 * d(B, E)) + (.25 * d(C, F))$. $D(u, v)$ is used to correlate similar words in the linear problems, which is shown in the next sections. Note $D(u, v)$ satisfies the identity and triangle inequality. $D(u, v) = D(v, u)$ and $D(u, v) + D(v, w) \leq D(u, w)$.

3.1 Learning word probabilities from examples

Empirical word probabilities worked very well on recovering the training set, but performed very poorly at computing unobserved word probabilities (shown in chapter 4. An LP, linear problem, can incorporate word similarity to improve computing unobserved word probabilities. A set of proteins with known structure is used to create an LP, which can then be passed to an LP solver. This section outlines the inequalities added based on the observed training set. First, an inequality is added for every observed word to restrict the sum of the structure probabilities to 1.

$$P_E(w) + P_H(w) + P_L(w) = 1 \tag{3.2}$$

3.1.1 Coherence inequalities

Then evolutionary similar words are restricted in their probability difference to being at most a fraction of their evolutionary distance, equation 3.1. These inequalities are called coherence inequalities. The fraction was an input into the LP termed ξ.

$$\begin{aligned} P_s(w) - P_s(v) &\leq \xi \ D(w, v) \\ -(P_s(w) - P_s(v)) &\leq \xi \ D(w, v) \end{aligned} \tag{3.3}$$

This produces six inequalities per pair of words, two for each structure type, and one in-

equality for every word. Several methods were tried to determine how many pairs of words to

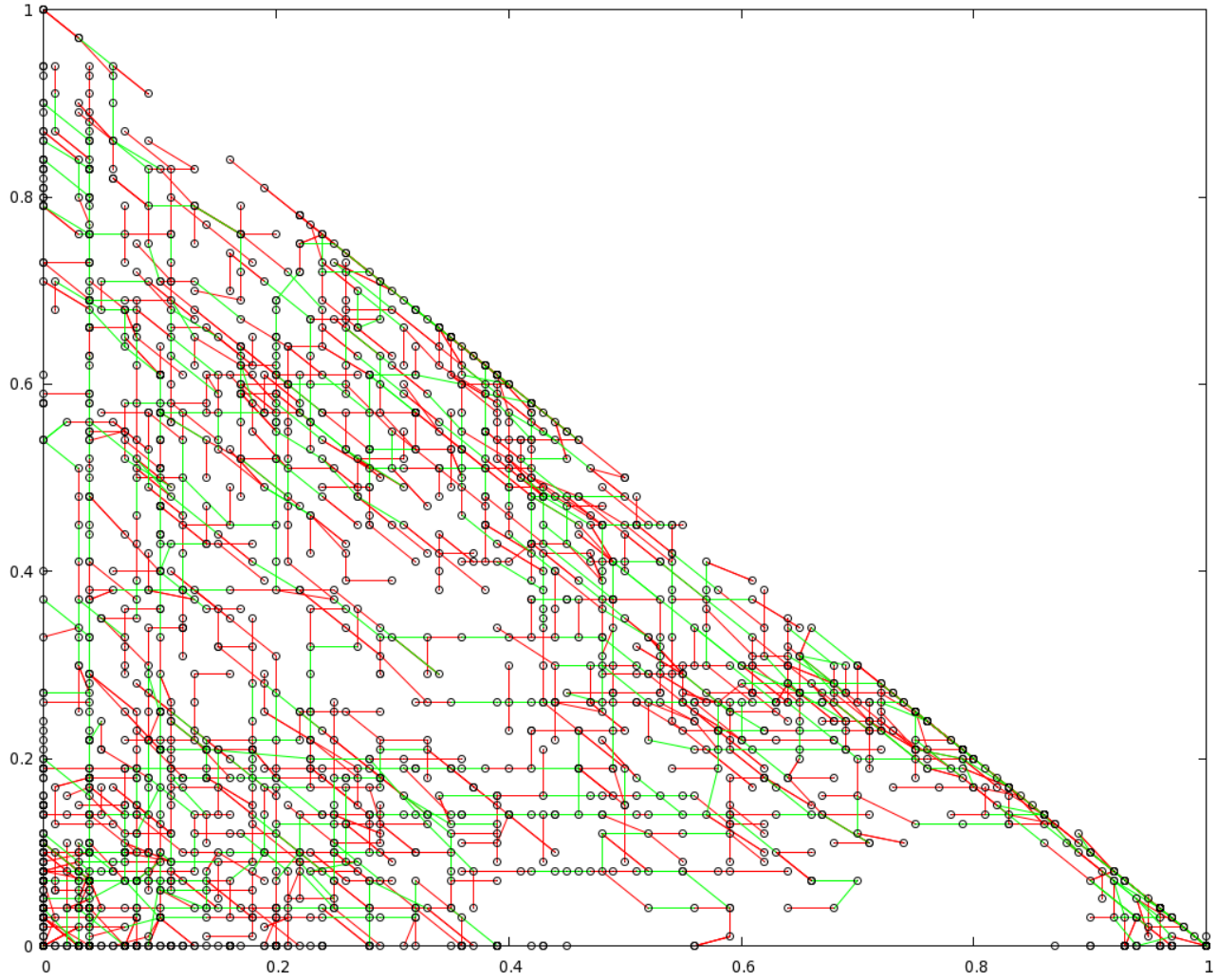

Figure 3.1: Each point is a word observed by the LP. Red lines are the k closest words. Green lines are the l spanning trees

produce inequalities for. The final method creates a spanning tree among all observed words using evolutionary distance, $D(v, w)$, where coherence inequalities, from above, are added to the LP for every edge. First, every word is paired with the k closest words. This forms a sparsely connected graph. Then edges are added to the graph to form a minimum spanning tree using

Dijkstra's algorithm. The minimum spanning tree is computed l times, each spanning tree using independent edges. k and l were both inputs to the LP. The coherence inequalities are the same for the LP's for all the prediction methods. Figure 3.1 shows a graph of word probabilities connected by edges, which represent the coherence inequalities that were added to the LP. In this graph k and l equal 1.

3.1.2 Margin inequalities

Margin inequalities are added to the LP's for every position in the training set to estimate word probabilities. The margin inequalities for the word averaging prediction method were:

$$\begin{aligned} g(i,s) &\geq g(i,s') + \delta - e_i \\ g(i,s) &\geq g(i,s'') + \delta - e_i \end{aligned} \tag{3.4}$$

$g(i,s)$ is equation 2.1. s' and s'' are the incorrect structures. δ is a global variable in the LP that is maximized, and e_i is a position specific variable accounting for error. These equations try to make the difference between the correct prediction and the incorrect predictions as large as possible. For the context averaging prediction method $g(i,s)$ is just replaced with $F_S(i)$.

$$\begin{aligned} F_{\mathbf{S}}(i) &\geq F_{\mathbf{S}'}(i) + \delta - e_i \\ F_{\mathbf{S}}(i) &\geq F_{\mathbf{S}''}(i) + \delta - e_i \end{aligned} \tag{3.5}$$

Here the sum of all contexts with the correct structure are push as far away from the sum of the contexts with the incorrect structure. For the dynamic prediction method the following margin inequalities are used:

$$
\begin{aligned}
R(i,c) &\geq 1/(CL+1)^2 \; R_{\mathbf{S}'}(i) + \delta_i \\
R(i,c) &\geq 1/(CL+1)^2 \; R_{\mathbf{S}''}(i) + \delta_i
\end{aligned}
\tag{3.6}
$$

$R(i,c)$ is equation 2.5. $R_{\mathbf{S}'}$ is the sum of $R(i,c)$ across all contexts with s' at its center. $R_{\mathbf{S}''}$ is the same except with s'' at the center. δ_i is the difference between the other variables which is maximized in the LP, and the error variable was not needed. These margin inequalities try to make the correct prediction greater than the average of the incorrect predictions. The average is used because inserting an inequality for every incorrect context created too many inequalities.

3.2 Computing word probabilities for new sequences

Prediction on a new sequence is likely to encounter words that were not observed in the LP. Their probability is interpolated from the k closest words that were observed by the LP. The interpolation method computes a weighted average based on the word's distance to the unobserved word.

$$
P_s(q) = \sum_{w \in k} \lambda_w \; P_s w \tag{3.7}
$$

$$
\lambda_w = \frac{\frac{1}{D(q,w)}}{\sum_{v \in k} \frac{1}{D(q,v)}}
$$

q is the unobserved word and k is the set of closest words. A naive approach to finding the k closest words would require scanning the entire set of observed words and be very costly. A clustered structure on the set of observed words is created to reduce the cost. A cluster is a multidimensional tree. Each internal node contains a representative and the maximum distance between the representative and any of its descendants. The number of children per node is

variable. Every leaf node is a single word. This structure allows sub-clusters to be pruned, skipped, during a search of the cluster.

While searching the cluster, a list of the k closest words encountered so far is maintained. A sub-cluster rooted at node v can be pruned if the distance from the current unobserved query word, q, to the representative of v minus the maximum distance from v to any of its children is greater than or equal to the distance from the query word to the current kth closest. If $D(q, v_{rep}) - v_{max} >= D(q, cl_k)$ then prune v, where v_{rep} is v's representative, v_{max} is v's maximum distance, and cl_k is the current kth closest word.

The cluster is searched recursively. At each node, its children are ordered by their distance to the query word, shortest first. The children are then searched recursively in order if they cannot be pruned. This way the search will attempt to find the closest words first so other portions of the cluster can be pruned.

3.2.1 Building the cluster

Finding a good tree structure for the cluster is essential to increase pruning and decrease the search time, and creating a cluster can be done off line. The cluster is created recursively. At each node the list of descendants is searched for the two farthest away from each other, doing an N by N comparison. These two words become temporary representatives for the first two children. The next child's representative is selected as the one farthest away from the current representatives. After M representatives have been selected, each descendant is assigned to the child with the closest representative.

For each child a new representative is chosen by finding the word with the smallest maximum distance to all other words. The process is repeated again, reassigning descendants to a child and computing new representatives, until the maximum distance for each child stops decreasing.

This was chosen as a stopping point because small maximum distances will increase pruning. M, the number of representatives, starts at a low threshold and is increased by 1 until it reaches the upper threshold or the average overlap between children begins to increase. The overlap between two children equals the sum of the maximum distances divided by the distance between the representatives minus 1. $(v_{max} + u_{max})/D(v_{rep}, u_{rep}) - 1 = overlap$. This is computed for every pair of children and divided by the total number of pairs to get the average overlap. This process is started at the root and repeated for every child until the number of words or max distance drops below a set threshold.

Chapter 4

Experimental Results

4.1 Setup for experiments

The Astrol 40% [3] data set was used for all experiments. It is a subset of the Protein Data Bank [2] where no two proteins share 40% identity, providing a training set with a wide variety of proteins and even distribution. STRIDE [5] was used to convert the tertiary structure in the protein data bank files into secondary structure. STRIDE outputs eight structure types, which was reduced to three using a common CK mapping [4]: $E \rightarrow E, H \rightarrow H$, all others $\rightarrow L$. The Blosum62 [6] evolutionary matrix was used for computing distances between words. CBSM 60 [10] was also tried, but it produced slightly lower accuracy. Several reductions on the set of amino acids were tried, but the full set achieved the best results. GLPK, which is part of the GNU project, was used primarily and ILOG CPLEX was used, thanks to the Systems and Industrial Engineering department, for solving the LPs. The data set was divided into fractions for training and testing. Typically 1/10th and 1/5th of the data set was trained on and the remainder was used for testing. Larger trainings sets generated LP's too large to solve.

Words of size 3 and 5 were used mostly. Using word size 3, solving the LP took on the order of days, and word size 5 increased the running time to over a week. Word sizes larger than 5 have a very large domain of possible words. With very large domains, the number of

observed words is a small fraction of the domain, causing issues with interpolating unobserved word probabilities.

Word Size	Window Length	Training Set			Recovery		
			OBJ	ξ	Test	Train	Time
3	7	10	0.19	0.1	58.9%	73.5%	8d 0:10
3	7	10	0.24	0.15	58.6%	75.4%	4d 22:00
3	7	10	0.28	0.2	57.6%	77.1%	2d 17:14
3	7	10	0.32	0.3	55.7%	79.5%	1d 16:08
3	7	10	0.34	0.4	54.4%	80.4%	7d 11:30
3	7	10	0.36	0.5	53.4%	81.1%	1d 2:05

Figure 4.1: This table shows the accuracy of the training and test sets in relation to varying ξ using the word averaging prediction method.

4.2 Results

In general, all methods were able to achieve around 58% accuracy on the test set. Accuracy percentage is the percentage of predictions that matched the known secondary structure, one for every position in a protein sequence. Many variations to the LP's and prediction methods were tried. Figure 4.1 shows accuracy results on the training and test sets in relation to varying ξ, using the word averaging prediction method. 1/10th of the data set was used for training and 9/10ths was used for testing. OBJ stands for the objective value of the solved LP, which equals the value of δ minus the average error from equation 3.4. In other words, it is the average amount of distance between the correct prediction and the incorrect prediction. The time column shows the amount of time it took to solve the LP in days, hours, and minutes. One important thing to notice is the relationship between ξ and the accuracy percentages. As ξ decreases, the test accuracy increases and the training accuracy decreases. This shows that the

coherence inequalities, equation 3.3, help with interpolating unobserved word probabilities, but only to a certain point.

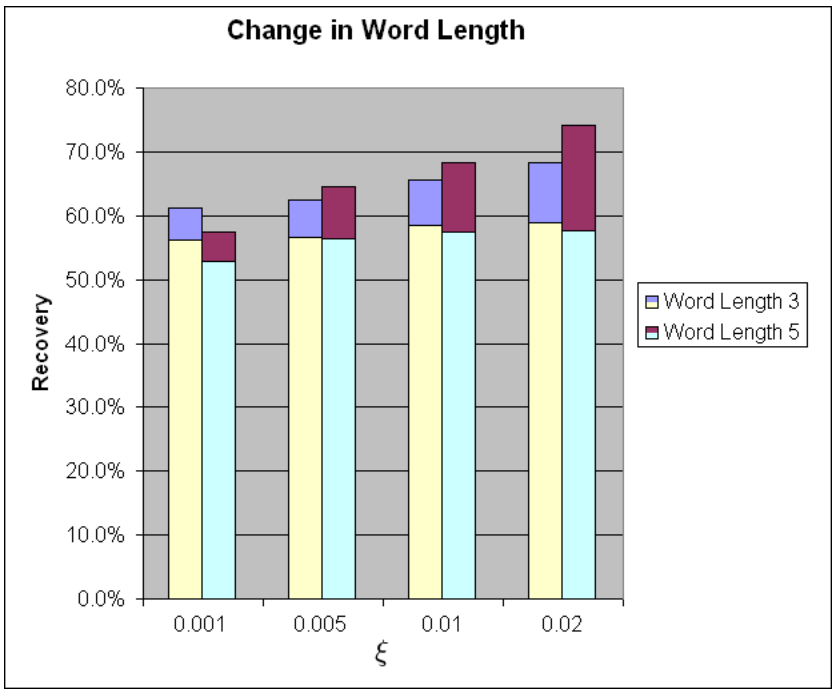

Figure 4.2: Accuracy of word length 3 verses 5 with variations in ξ using the context averaging prediction method. Each bar shows the accuracy of the test(lower value) and training(higher value) set. Y axis is accuracy percentage. X axis is ξ value

Figure 4.2 shows the difference in accuracy by using word length 3 and 5 using the context averaging prediction method. A context length was 4 and 1/10th of the data set was used for learning. With word length 5, the accuracy on the training set when up, but the accuracy on the training set went down, which was unexpected. Investigation revealed this was due to the change in the number of possible words. Since they both used the same training set, roughly the same number of words was observed. That number was a larger fraction of the domain of possible words for word length 3 then for word length 5. This made interpolation of unobserved words for word size 5 more difficult and lowered the accuracy on the test set. Increasing the

Word Size	Context Length	Traing Set	OBJ	ξ	Recovery		Time
					Test	Train	
3	4	10	0.36	0.01	57.0%	64.4%	2d 21:36
3	4	10	0.46	0.02	57.3%	68.9%	3d 19:09
3	4	10	0.52	0.03	57.2%	71.8%	3d 2:06
3	4	10	0.56	0.04	56.6%	74.6%	3d 1:54
3	4	10	0.59	0.05	56.0%	76.4%	4d 2:54
3	4	10	0.71	0.1	53.6%	82.8%	3d 1:26
5	4	10	0.33	0.0005	54.2%	62.8%	6d 13:10
5	4	10	0.44	0.01	55.1%	68.9%	15d 23:23
5	4	10	0.58	0.02	54.9%	77.8%	21d 18:14
5	4	10	0.82	0.05	53.4%	91.8%	7d 21:59

Figure 4.3: This table shows the results for the dynamic prediction algorithm. OBJ equals the average δ, separation, between the correct context and the average of the incorrect contexts. Time is in days, hours:minutes.

training set vastly increases the time to solve the LP, making larger training sets for word length 5 infeasible.

Figure 4.2 shows the results for the dynamic prediction. The accuracy of the training set is more sensitive to ξ than with the word averaging prediction method, dropping very quickly, but also higher at larger values of ξ. Using word size 5 high accuracy on the test set was achieved, 91.8% with ξ equal to .05. The time to solve the dynamic prediction LP with word size 5 was very long, limiting the amount of results. As noted earlier, the limiting factor on accuracy of the test set was interpolating unobserved words.

In an effort to evaluate the prediction methods apart from computing word probabilities, they were tested with empirical probabilities from the entire set. The dynamic prediction method achieved over 99% accuracy using empirical probabilities. To futher test the resiliency of the dynamic prediction method, it was given the perfect word probability at every position minus

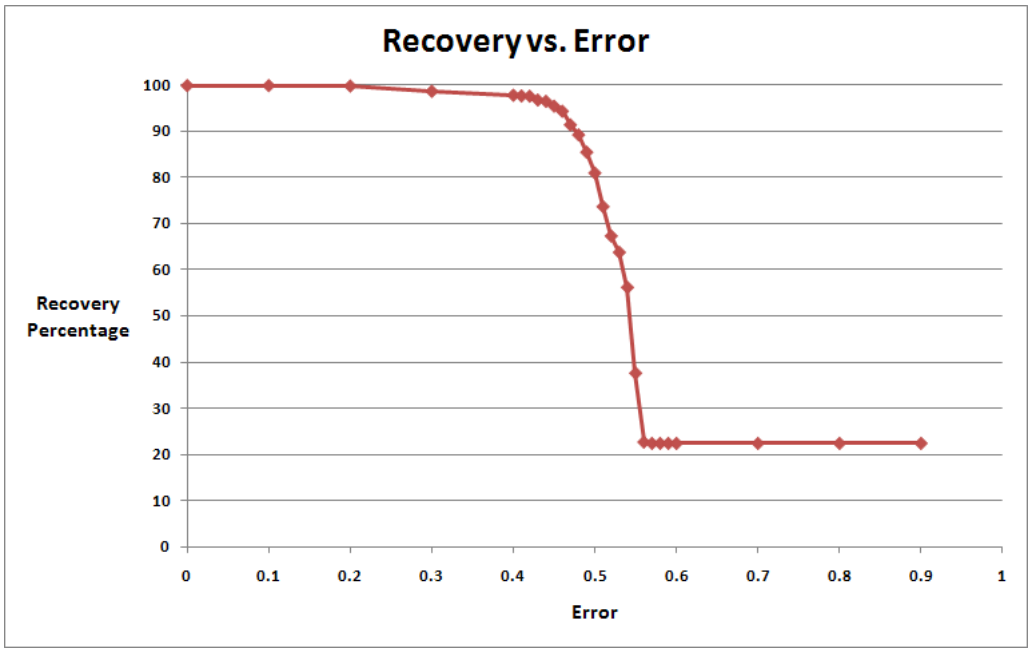

Figure 4.4: This plot shows the dynamic prediction algorithm's accuracy percentage verses the amount of error introduced into the word probabilities.

some error. Perfect word probability means the probability for the correct structure equals 1, and the probability for the incorrect structures equals 0. For example, if the correct structure for word w was E, then $P_E(w)$ would equal 1 and both $P_H(w)$ and $P_L(w)$ would equal 0. Error, e, was added to the probabilities by subtracting e from the correct structure, and distributing it evenly amoung the incorrect structures. Figure 4.2 shows a plot of the accuracy percentage verses the amount of error induced. The accuracy does very well and begins to decline at .4 error, and then drops steeply at .46 error. It stops declining at 22% because the algorithm begins predicting the same structure at every positon.

This shows that the dynamic prediction method can perform well with a large amount of error, up to 40%, and reveals that interpolating probabilities for unobserved words is very difficult. Interpolation of unobserved word probabilities was tried using empirical data from 1/2

of the data set. The interpolated value was then compared with the empirical probability from the entire set. On average the interpolated probability was off by .57, using Euclidean distance between the three stucture probabilities. As shown above, this is more error than the prediction algorithm can handle. As seen in figure 4.1, interpolation worked better on word probabilities computed from the LPs, but the ability to recover the test set decreased drastically.

Chapter 5

Conclusions and Future Research

The results show that the dynamic prediction method can achieve very good accuracy, but the interpolation method for probabilities on unobserved words does not perform well. There are several possible explanations. Words may have to be within a certain distance in order for there to be any similarity. This would have several implications. First, the coherence inequalities, equation 3.3, would be overly restrictive if the words are very different. There could be a subset of words that are not very close to any other observed words, which is possible due to the high dimensionality of the distance function. Changing ξ to a non-linear function, e.g. $\xi(x) = x^2$, might help account for this. Second, the training set might be too small, causing the observed words to have weak correlations to one another. This would cause problems in learning good probabilities from the LP's and trying to interpolate unobserved word probabilities. Large training sets drastically increase the time to solve an LP, limiting the training sets to 1/5th of the entire data set, so small LP formulations would need to be developed. An attempt to reduce the LPs was tried by using three variables per run of a structure, instead one for every position. However, this actually increased the time to solve the LP.

It is also possible the distance function, equation 3.1, is too coarse. For example, two words that are off by 1, $ABCDE$ and $BCDEF$, would have a high distance because each pair of

characters is different, yet the words are very similar. Word alignment distance could be used to get more accurate distances. It is also very possible evolutionary distance is a poor metric for structure similarity between words. This indeed begs the question what is a good metric and could require much trial and error to answer.

The next major addition to the prediction methods is to include profiles. A profile contains several proteins with high similarity to the target protein. When predicting a position in the target sequence, i, all the words in the profile sequences at position i could be taken into account. This would automatically provide a set of similar words, with a high chance that some of them are observed. This could greatly improve the interpolation method and boost the accuracy. Other prediction methods such as PSIPRED [7] report a 10% gain in accuracy by using profiles. The drawback to using profiles is the increased prediction time. Profiles require a database search per protein sequence, which has inherent costs.

Bibliography

[1] Altschul,S.F., Madden,T.L., Schäffer,A.A., Zhang,J.H., Ahange,Z., Miller,W. and Lipman,D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, **25**, 3389-3402.

[2] Berman,H.M., Westbrook,J., Feng,Z., Gilliland,G., Bhat,T.N., Weissig,H., Shindyalov,I.N. and Bourne,P.E. (2000) The Protein Data Bank. *Nucleic Acids Res.*, **28**, 235-242.

[3] Brenner,S.E., Koehl,P. and Levitt,M. (2000) The ASTRAL compendium for protein structure and sequence analysis. *Nucleic Acids Res.*, **28**, 254-256.

[4] Crooks,G.E., Brenner,S.E., (2004) Protein secondary structure: entropy, correlations and prediction. *Bioinformatics*, **20**, 1603-1611.

[5] Frishman,D. and Argos,P. (1995) Knowledge-based protein socondary structure assignment. *Proteins*, **27**, 329-335.

[6] Henikoff, S. and Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA* **89**: 10915-10919.

[7] Jones DT. (1999) Protein secondary structure prediction based on position-specific scoring matrices. *J. Mol. Biol.* **292**: 195-202.

[8] Rumelhart,D.E., Hinton,G.E. and Williams,R.J. (1986) Learning representations by back-propagating errors. *Nature*, **323**, 533-536.

[9] Szent-gyorgyi,A.G. and Cohen,C. (1957) Role of proline in poly-peptide chain configuration of proteins. *Science*, **126**, 597-698.

[10] Liu, X., Zheng, W.M., Prediction of protein secondary structure based on residue pairs. *J. Bioinformatics Computational Biol. 2004*, **2**, 343352.