Spring 2011

AB INITIO PROTEIN STRUCTURE PREDICTION ALGORITHMS

Maciej Kicinski
San Jose State University

AB INITIO PROTEIN STRUCTURE PREDICTION ALGORITHMS

A Working Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Maciej Kicinski

May 2011

The Designated Thesis Committee Approves the Working Project Titled

AB INITIO PROTEIN STRUCTURE PREDICTION ALGORITHMS

by

Maciej Kicinski

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2011

ABSTRACT

AB INITIO PROTEIN STRUCTURE PREDICTION ALGORITHMS

by Maciej Kicinski

Genes that encode novel proteins are constantly being discovered and added to databases, but the speed with which their structures are being determined is not keeping up with this rate of discovery. Currently, homology and threading methods perform the best for protein structure prediction, but they are not appropriate to use for all proteins. Still, the best way to determine a protein's structure is through biological experimentation. This research looks into possible methods and relations that pertain to *ab initio* protein structure prediction. The study includes the use of positional and transitional probabilities of amino acids obtained from a non-redundant set of proteins created by Jpred for training computational methods. The methods this study focuses on are Hidden Markov Models and incorporating neighboring amino acids in the primary structure of proteins with the above-mentioned probabilities. The methods are presented to predict the secondary structure of amino acids without relying on the existence of a homolog. The main goal of this research is to be able to obtain information from an amino acid sequence that could be used for all future predictions of protein structures. Further, analysis of the performance of the methods is presented for explanation of how they could be incorporated in current and future work.

Table of Contents

List of Figures

I. Background

i. Introduction

Proteins are essential parts of organisms and participate in virtually every process within a cell. A protein's structure is vital to understanding what the protein's function is. A protein's structure can be determined experimentally through processes like x-ray crystallography, but these methods are slow compared to how quickly new proteins, without determined structures, are being discovered. The field of protein structure prediction has been around since the 1960s. It aims to predict structures for proteins which have not been determined experimentally. With more understanding, protein structure prediction could be used as a base for creating medication for dysfunctional proteins in which the medicine could shut off the protein, or even make it work properly. This could also lead to treatment of genetic diseases and extending telomeres to extend life.

ii. Biology

DNA and RNA are made up of sequences of four different nucleotides: adenine(A), cytosine(C), guanine(G), and thymine(T) for DNA and uracil(U) replaces thymine for RNA. A stretch of DNA that represents a protein is called a gene. DNA is transcribed by polymerase into RNA. RNA is then translated into amino acid sequences by ribosomes. Triplets of nucleotides, called codons, are read by the cellular machinery to encode for a single amino acid, which are sequentially bonded together as the codons are translated to form a protein. There are twenty different amino acids, which are encoded for by the sixty-four different triplet codon possibilities that can be made from four nucleotides. Three of these codons represent a special stop code that tells ribosomes, when to end translation of the RNA. This leaves sixty-one codons for twenty amino acids which means that most amino acids are encoded by more than one codon, meaning the amino acid code is redundant. Amino acids chain together into a sequence to form a protein which eventually takes on a three-dimensional structure that influences their abilities to interact with other proteins and perform various cellular functions.

Protein structure has four levels that define several aspects of protein structure: the primary structure, the secondary structure, tertiary structure, and quaternary structure. The primary structure of a protein is simply the sequence of amino acids that is translated from a messenger RNA(mRNA). The secondary structure of the protein is sub-regions of the primary structure that begin to interact and form alpha-helices(α-helices) and beta-sheets(β-sheets). The tertiary structure results when α-helices and β-sheets within a single protein to form a three-dimensional shape. The final level of protein structure is quaternary structure, which refers to when more than one protein come together to form a complex. An example of a quaternary structure is human hemoglobin, which is made up of four distinct subunits, each an individual chain of amino acids, but functions as a single complex. A protein's final structure is deteremined by its inherent properties and when it becomes stable in a low energy state. In some cases, a chaperone helps another protein fold by introducing a lower energy barrier and shapes the protein into a conformation that the protein would not take on itself under physiological conditions.

The combination of the properties that allow a specific protein to form into a certain structure is not completely known. There are many inherent properties that amino acids have that are involved in determining the structure of a protein. One of the most important distinguishing factor of amino acids is their different tails which are also called the R Groups. Other factors play key roles in determining the final structure of a protein, these include: the energy level of the structure which needs to be low and stable and links between amino acids like sulfide bonds. These are not the only factors and there could even be properties that are not known yet with the current understanding of biology. Even just a single amino acid has several properties to consider, so that a chain of them has a perpetual encyclopedia of knowledge to consider.

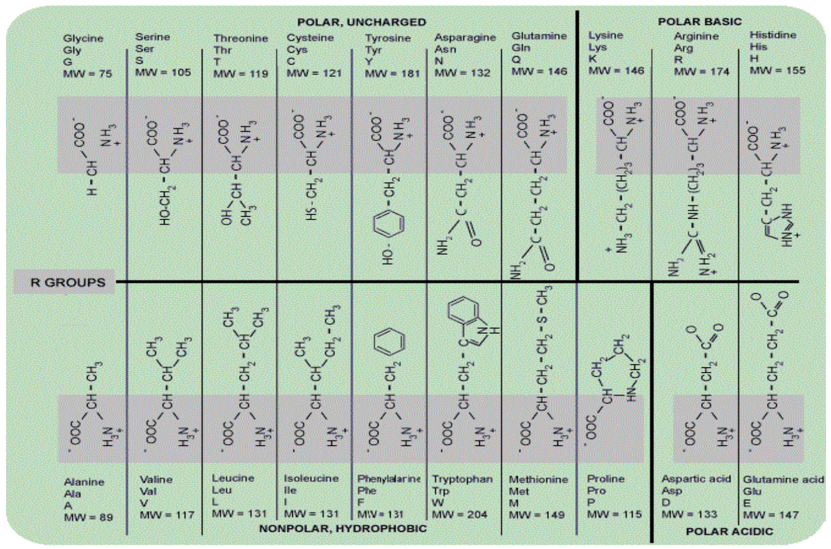


Figure 1. Amino Acids Chart. The twenty amino acids, grouped according to the character of their side chain or R group. This figure was slightly edited for display purpose from the original that is listed as reference. (The Biotechnology Project, 2011)

Figure 1 depicts the chemical differences between amino acids. All amino acids have a C-terminus, with a carboxyl group (COO-), and an N-terminus, with an amino group (NH3). The C-terminus and N-terminus are shown in the figure in gray. The R groups are unique parts of each amino acid located at the center of the figure. What individually distinguishes all of the amino acids is the different 'tails', which are the molecules that distinguish the different amino acids. Many of these tails only slightly differ from one another and yet their contribution to the final conformation of a protein can be huge. A single amino acid change can alter a protein's structure so completely that it is no longer functional. The amino acids are separated into four groups based on the chemical properties of their unique side groups: non-polar, polar, basic, and acidic. Non-polar and polar also are called hydrophobic and hydrophilic in regards to how hydrophilic is attracted to water while hydrophobic is repelled by water. One key property of hydrophobic and hydrophilic amino acids is that hydrophobic amino acids are more likely to be found on the inside of a protein structure while hydrophilic amino acids are on the outside of the structure since cells are watery environments.

The average human body contains around 100,000 proteins with the longest proteins being over 30,000 amino acids long (Alberts, et al., 2004). At first, it was believed that the human body had 100,000 genes, meaning that one gene would code for one protein, but recently, advances in human genomic research demonstrated that genes could code for more than one protein. This finding lowered the estimated number of genes in the human body to between 20,000 and 25,000. The 100,000 proteins in each human all play a part in defining that person. They are essential parts of organisms and participate in virtually every process within a cell and between cells. Some of the functions of proteins include: enzymes that catalyze biochemical reactions, structural or mechanical functions, maintaining cell shape, cell signaling, immune responses, cell adhesion, and regulating the cell cycle (Alberts, et al., 2004). To be able to understand protein structure can lead to better medicine, treatment for genetic diseases, and perhaps longer life.

The difficulty of predicting protein structure may be better understood with an example like human hemoglobin and sickle sell anemia. Human hemoglobin is a complex of four proteins with a globular structure designed to carry oxygen and travel through blood vessels. The disease sickle cell anemia is caused by a single mutation in one hemoglobin subunit that drastically alters the protein structure. In this case, a single nucleotide change alters the codon sequence and causes a hydrophobic amino acid to be incorporated into the amino acid chain in place of a hydrophilic one.

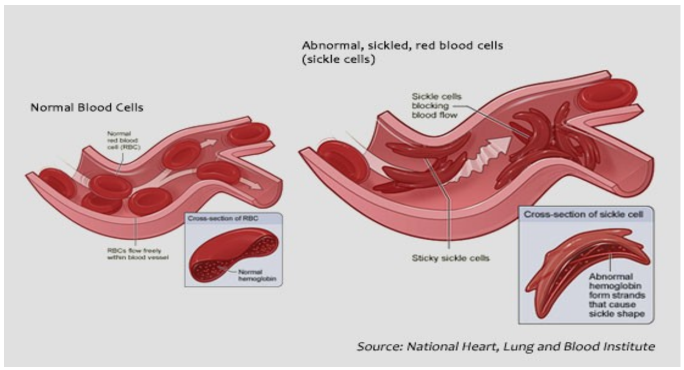

Abnormal, sickled, red blood cells (sickle cells)
Normal Blood Cells
Normal red blood cell (RBC)
Sickle cells blocking blood flow
Cross-section of RBC
RBCs flow freely within blood vessel
Normal hemoglobin
Sticky sickle cells
Cross-section of sickle cell
Abnormal hemoglobin form strands that cause sickle shape
Source: National Heart, Lung and Blood Institute

Figure 2. Regular and Mutated Hemoglobin. The final quaternary structure of regular blood cells and mutated blood cells, Sickle Cell anemia, as they travel through a blood vessel.

Normally, the final structure of red blood cells are quarternary globular shape formed by hemoglobin proteins, but the change from a hydrophilic amino acid to a hydrophobic amino acid causes the resulting hemoglobin proteins to chain together in a line (a sickle shape), which then does not flow through blood vessels as well. Two different amino acids sequences, which are the same length and only differ by one amino acid produces result in two completely different final protein conformations. The structures describe above were obtained through x-ray crystallography and are used to show how difficult a task protein structure prediction has. Without a completely accurate protein structure prediction, the resulting activity of the protein to be determined could be drasticly changed.

iii. Protein Structure Prediction

Protein structure prediction began in the 1960s, but one of the earliest influential prediction methods, called the Chou-Fasman Method, was introduced in the mid-1970s(Clair and Visick, 2010). The Chou-Fasman Method was designed for predicting secondary protein structure and used parameters that were derived from the few protein structures determined experimentally at the time. For a single protein secondary structure, the method could achieve accuracy in the range of 50%-60%. Since then, many more algorithms and methods have been developed for protein secondary structure prediction. Current top secondary structure prediction tools boast accuracies of greater than 80% (Cole, et al., 2008). There are also tools introduced for the prediction of tertiary structure and the prediction of quaternary structure of proteins.

Currently, protein structure prediction methods can be categorized into three different types of modeling methodologies: Homology modeling, Threading, and *ab initio*. Homology modeling prediction revolves around finding a homologous protein, a to the protein whose structure is to be predicted. A homolog shares a common ancestor and, even though time may have changed the exact protein composition, they can be very similiar in structure and function which is generally found by locating conservered regions where the amino acids in the protein are the same across multiple species. The structure of the homologous protein can be used to model the structure of the novel protein under the assumption that the two proteins would share a similar structure based on their shared amino acid sequences. If no homologous protein has been identified, then homology modeling is impossible to use for predicting the structure of a novel protein. Threading methods can be used to make predictions about protein structure in this case. Threading looks for several proteins with the small stretches of similar sequences that make up the sequence of a novel protein rather than concentrating on finding a homolog. Threading uses whole databases to find similar sequences from several different proteins for use in predicting a particular protein. Like homolog modeling, threading only works if there are identified proteins with similar sequences and determined structures. If neither of these methods is possible, then ab inito methods of preotein structure prediction need to be developed.

The latin term '*ab initio*' is used in bioinformatics to describe methods used to predict protein structures without the benefit of homologs or any other information about the structure of a protein. With respect to protein structure prediction, *ab initio* means without prior knowledge. The goal of *ab initio* protein structure prediction is to predict a protein's structure accurately by focusing on the chemical and physical properties of the amino acid sequence making up the mature protein.

There have been many methods, algorithms, and tools developed for the predicting of protein structure and there are always new tools and enhancements to current tools being worked on. An excellent site to follow is the ExPASy Proteomics Server (ExPASy, 2011). Some notable tools, that are included on that server, are Jpred and PSIPred which both use a position specific scoring matrix(PSSM), which contains position values based on similiar sequences, that is retrieved from PSI-BLAST, a sequence alignment tool that generates PSSMs. Jpred's newest revision, Jpred3, incorporates Jnet 2.0 the highest prediction accuracy for secondary structure at greater than 81% (Cole, et al., 2008). The two most popular databases for protein structure are the Protein Data Bank (PDB) and the NCBI Protein Database. The biggest obstacle to improving prediction tools in general is still the slow pace of experimental advancements in biological and biochemical researchStill, new protein structures are constantly being determined, increasing the data available to refine protein structure prediction methods, which will eventually lead to a breakthrough in the field.

In addition to tools that attempt to directly predict the structure of the protein using algorithms and other information, another innovation in prediction technology has been to employ unused processing time on thousands of personal computers. People opt to download a prediction program that, when their computers are idle, uses brute force calculations to model all possible different folds of proteins and measure the energy state of the predicted fold to find the best possible folds (Simons, et al., 1999). The computational time of the procedure is tremendous even with a great number of people allowing the program to run all the time. Another interesting tool that was created was FoldIt (foldit, 2011), a protein folding game for the computer. Rather than predict a protein fold, the tool measures the stability of a particular protein structure as a person plays around by moving, assigning structure, and contorting the protein sequence on the screen. The person's final result is then scored by how low they can get the free energy of the protein.

As of this time, tools that make use of homology modeling and threading achieve the best results in secondary and tertiary structure prediction. However, within some genomes more than 50% of proteins lack a structural homologue that is similar enough to the query sequence to build a confident model for prediction (Cole, et al., 2008). The only alternatives are either to use an *ab initio* approach, which is not as accurate as the other two methods, or to determine the structure through laboratory experiments. The focus of my project is to improve *ab initio* protein structure prediction by attempting to find some interesting correlation that appears in all amino acids and protein structures or to directly improve another algorithm by some means.

To understand better protein structure prediction, it is worth knowing how the accuracy of a particular method is measured. The most common measure for how well a tool performs is the Q3 score, which involves computing the number of times the tool rates α-helices, β-sheets, and coils correctly compared to the protein's determined secondary structure. A tool or method may state that it has 80% accuracy, or may say that it achieves a Q3 score of 80%. The terms are essentially interchangeable in protein secondary structure prediction, but it is more common to make such a statement using the Q3 measurement. The goal of protein secondary structure prediction is to be able to achieve a Q3 score of 100% or one such that a protein's final conformation can be determined, which can be used to direct further experiments to confirm the predicted structure and to determine the protein’s function.

The hope of proteomics, the field relating to proteins, is to be able to know a protein's final structure from its primary sequence, its amino acid sequence, and then that would also bring knowledge to the protein's function. Proteins appear in life in performing almost every imaginable type of action. Being able to understand proteins and construct proteins would be able to help people everywhere. With that type of information, the possibilities that would open to the medical field would be endless. In the next section, the process with which was taken to undergo implementation of the working project is discussed.

II. Literature Search

Exploring Bioinformatics (Clair and Visick, 2010) is a very recent text it has up to date information in the field. The book has good insight into protein structure prediction with a chapter dedicated to RNA and Protein Structure Prediction that includes a project involving the Chou-Fasman Secondary Structure Prediction method. The Chou-Fasman method presented in 1974 was one of the first methods on the subject and it was also my starting point for the implementation of protein secondary structure prediction. The method involves a matrix of two values: propensity values, the likelihood that a given amino acid will appear within the structure, and frequency values, the frequency with which a given amino acid is found in a hairpin turn. Taking these values into account the method then predicts regions of α-helics, regions of β-sheets, and positions where β-turns may appear.

After implementing Chou-Fasman in Perl, ExPASy Proteomics tools website was a great place to view current Bioinformatics tools for protein studying. There are a great number of tools listed on the site, so my search was limited to those involved in secondary structure prediction, which included an online implementation of the Chou-Fasman method, and a few newer tools including Jpred. With no clear indication which tool was superior, an experiment was run with all the secondary structure prediction tools that were accessible. The test involved using each tool to predict a known protein, the human beta-globin (HBB), and then look into the tool that scores the best. Jpred and Jufo tied with over ninety percent of the secondary structure of HBB predicted correctly with both using similiar sequences in neural networks. Jpred became an obvious choice to use as a next step since information regarding its design was easily accessible.

Jpred is currently on its third revision, which includes the second revision of Jnet, Jpred's specialized method for secondary structure prediction. Jpred's first presentation was as a consensus method which involved taking predictions from different tools and taking the predictions that appeared the most. With the development of Jnet, Jpred turned into a method involving position-specific scoring matrices and homology. The Jpred website had the training and testing sets that were used in the training of Jnet1, making Jpred an exceptional choice to follow up in this project. The training and testing sets had been optimized and had had redundancy removed. Being able to know a protein's structure based on another protein can be helpful, but if there are no known proteins relating to that protein, then another approach is needed. This led me to attempt to try and find new information that *ab initio* protein structure prediction might be able to use for any type of protein structure prediction. In the next section, literature pertaining to protein structure prediction that is referenced by this paper is reviewed.

Clair C. S. and Visick J. (2010) in their chapter on RNA and Protein Structure prediction state that because viruses replicate through the use of hijacking other cells, viral infections present special challenges for the development of medicine. Despite decades of trying, there are few effective antiviral drugs and no true cure for any viral disease. A new understanding of the three-dimensional structure of virus proteins may be the way to be able to solve this problem which is just one of the feats that could be achieved through a better understanding of protein structure and protein structure prediction.

Clair C. S. and Visick J. (2010) compare the number of protein sequences being discovered today to the effectiveness of protein structure prediction in which the current state of protein structure prediction can not compete with the speed that new sequences are being discovered. They explain that the standard method for experimentally determining the structure of a protein is X-ray crystallography and there are also newer methods, but they still cannot keep up to the rate at which proteins are being discovered.

Clair C. S. and Visick J. (2010) go on to mention that the number of novel proteins being stored in databases can lead to extensive analysis in the field of protein structure prediction and function. Our knowledge in biology is still a limiting factor, and that ability to predict tertiary structure of proteins is quite limited, which is unfortunate because a protein's function can sometimes be determined by its structure.

Clair C. S. and Visick J. (2010) discuss the structure of proteins and possible properties that can be used to determine how the protein's structure is formed. For example, β-sheets need to be held together by hydrogen bonding and, if they closely follow each other, it is considered a β-turn. The most important thing that they state is that proteins generally fold to reach their lowest energy state, which is also the proteins most stable state, but it is also affected by how the protein is synthesized.

Clair C. S. and Visick J. (2010) conclude by emphasizing the complexity of protein structures and how hard it is to predict a protein's structure with just its amino acid sequence stating that being able to do so could be considered the 'holy grail' of structural biology. Unfortunately, with our current understanding of biology, computational power and available tools are not good enough to be able to predict a protein's structure quickly. They state that a computer could take hours to simulate what happens in mere nanoseconds in nature.

Clair C. S. and Visick J. (2010). RNA and Protein Structure: Structure Prediction. *Exploring Bioinformatics: A Project-Based Approach, 1*(7), 197-230.

Alberts, et al. (2004) claim that we are all made up by our proteins. Protein functions can include: carrying messages from one cell to another, propelling organelles, providing immunity to infections, and even shaping other proteins. Most importantly, the authors state that the multiplicity of functions performed by proteins comes from the huge number of different shapes that they may adopt.

Alberts, et al. (2004) claim that the shape of a protein is specified by its amino acid sequence. The primary structure of a protein determines its secondary structure, which determines its tertiary structure, which determines what other proteins it will interact with and what cellular functions it will perform.

Alberts, et al. (2004) explain that different shapes of proteins depict their different functions by how the proteins are able to bind to other proteins or how small molecules allow them to perform an action. This allows proteins to accomplish tasks relating to transporting oxygen through the blood vessels as well as fighting off bacterial infections. Some proteins, called chaperones, have distinct functions of only folding other proteins into conformations that they would not normally take. Being able to understand proteins would lead us all to being able to understand medicine in a much more broad sense.

Alberts, et al. (2004) state that all proteins bind to other molecules and in this have a specific function to perform. In many proteins, the binding to another molecule is their only function. This can be the simple interaction of two proteins or an initial protein interaction could trigger a series downstream interactions that function to send signals throughout an organism.

Alberts, et al. (2004) end by stating that the large amount of data is increasing the speed with which the field of proteomics advances. Large-scale analysis in the field could bring us closer to understanding the fundamental concepts of living cells which would then be able to be used in many aspect from producing medicine to treat genetic diseases to being able to cure viral infections.

Alberts, Bray, Hopkin, Johnson, Lewis, Raff, Roberts, and Walter. (2004). Protein Structure and Function. *Essential Cell Biology, 2*(4), 119-157.

Chou P. Y. and Fasman G. D. (1974) computed α-helical, β-sheet, and random coil parameters, which they label as P_α, P_β, and P_c, for the 20 naturally occurring amino acids from the frequency of occurrence of each amino acid residue in α, β, and coil conformations in 15 proteins. A mechanism of protein folding is proposed where nucleation starts at the centers of helices, where P_α values are the highest, and propagates in both directions until helix breakers, lowest P_α values, terminate growth at both ends. For β-sheets, the case is the same except in regards to the P_β variable. The paper enables accurate prediction of protein secondary structure as well as provide insights into the next steps.

Chou P. Y. and Fasman G. D. (1974) state that the mechanism by which proteins fold, in terms of biological activity, has been a long sought goal. Some parameters were used that were recently discovered regarding amino acids favoritism to secondary structure types. For instance, Leu, Glu, and Ala were found most frequently in helical regions with Leu being the most prominent residue in inner helical cores. The paper presents an analysis of all 20 amino acids in 15 proteins with known structures and results in creating parameters to use in predicting secondary structure in other proteins.

Chou P. Y. and Fasman G. D. (1974) surveyed 15 proteins whose amino acid sequence and conformation were known through X-ray crystallography which they list. They tabulated the results of the listing amino acids residues in helical, β-sheet, and coil conformations in which they note that β-sheet residues differ slightly than findings from the time. The 2473 residues used from the 15 proteins should have more statistical reliability than earlier literature data including greater analysis of α-helical and β-sheet boundary regions.

Chou P. Y. and Fasman G. D. (1974) claim that the frequency of all 20 naturally occurring amino acids can be given in α-helical, inner α-helical, β-sheet, and coil regions and can be obtained when the occurrence of each conformation is divided by the amino acid's total occurrence in the 15 proteins. The following average values were calculated: average amino acids, helical residues, helical segments, residues, all per protein as well as other averages noting that since a relatively low number of proteins were studied that the averages could mean very little.

Chou P. Y. and Fasman G. D. (1974) note that there are several other interactions, like near-neighbor interactions and temperature, that have effects on protein that are not used by their algorithm. These parameters could give better results in secondary structure prediction if their exact roles were known however, by utilizing the P_α and P_β given, reasonable estimates of protein conformation could be made in a direct and simple manner.

Chou P. Y. and Fasman G. D. (1974). Conformational Parameters for Amino Acids in Helical, β-Sheet, and Random Coil Regions Calculated from Proteins. *Biochemistry, 13*(2), 211-222.

Jones D. T. (1999) explains the PSIPRED method as a two-stage neural network, taking multiple inputs and outputting several predictions which then are used together, based on position-specific scoring matrices generated by PSI-BLAST. Despite how simple and convenient the method is, the results are superior to other methods including a popular method called PHD. A testing set based on 187 unique folds and three-way cross-validation based on structural similarity was used as a criterion and resulted in average Q3 score between 76.5% to 78.3% which is the highest published score for any method to that day.

Jones D. T. (1999) explains that as a result of an influx of sequence data, interest in the ability to predict protein structure from an amino acid sequence has increased. Earlier methods for the prediction of secondary structure were based on either simple stereochemical principles or statistics. At present, the prediction of an unknown protein structure is best known by comparative modeling, but only when a suitable homologous protein can be found. The method described here makes use of neural networks, like PHD, and is greatly simplified, yet achieves a very high accuracy.

Jones D. T. (1999) split their prediction method into three stages: the generation of a sequence profile, prediction of an initial secondary structure, and filtering of the predicted structure. The main design goal of the mentioned method was to make it easily portable for use, both for the generation of sequence profiles and the actual prediction of secondary structure. PSIPRED uses the PSI-BLAST profiles directly eliminating the time-consuming multiple-sequence alignment stage. For testing purposes, in order to produce a more stringent testing set, proteins with similar folds to that of the training set were removed which left a testing set of 187 protein chains.

Jones D. T. (1999) states that the average Q3 score for the testing set was found to be 76% with a standard deviation of 7.8%. The average Solvent Accessibly score was 73.5% with a standard deviation of 12.7%. Using the simpler DSSP secondary structure, the by-residue average Q3 score was found to be as high as 78.3%. In participating of CASP3, the third protein structure prediction competion and using the most poorly predicting proteins, a Q3 score of 73.4% was achieved. On the same proteins, the most widely known algorithm, PHD, scored a Q3 score of 66.7%.

Jones D. T. (1999) concludes that it is not yet clear which factors contribute the most to the success of the PSIPRED method and work is currently underway to compare the results obtained from PSIPRED with those obtained from other methods. Jones D. T. (1999) goes on to suggest that the possible factors could include using pairwise local alignments, the use of iterated profiles, and testing in the laboratory. He states that the most significant conclusion that can be reached is that a very simple method is capable of producing excellent results.

Jones D. T. (1999). Protein Secondary Structure Prediction Based on Position-specific Scoring Matrices. *Journal of Molecular Biology, 292*(2), 195-202. doi:10.1006/jmbi.1999.3091

Cuff, Clamp, Siddiqui, Finlay, and Barton (1998) present an interactive secondary structure prediction server. They state that the server allows both single and multiple sequence alignment to be submitted that returns predictions from six different secondary structure algorithms. The prediction has a Q3 score of 72.9%. They also state that the server simplifies the use of the prediction algorithms and allows conservation patterns to be identified.

Cuff, Clamp, Siddiqui, Finlay, and Barton (1998) state that when predicting the secondary structure of a protein 'blind', meaning without threading or homology, that it is useful to use several prediction algorithms rather than just using a single one. They cite several examples in which combinations of methods have been successful, but combining those methods on a large scale is difficult because of the different inputs and outputs of the methods. Their solution was to develop flexible software that standardizes the input and output requirements of six prediction algorithms.

Cuff, Clamp, Siddiqui, Finlay, and Barton (1998) create a server that accepts a family of aligned protein sequences or a single protein as input types. They then run six different prediction methods which are: DSC, PHD, NNSSP, PREDATOR, and ZPRED. Those methods were chosen as they were the current state of the art prediction algorithms and they each used a different heuristic for their prediction.

Cuff, Clamp, Siddiqui, Finlay, and Barton (1998) main objective was to deploy a server that uses multiple protein secondary structure prediction algorithms using a standardized input and to output in a standardized way. The predictions and sequences are output and displayed in rendered HTML, Java, and POSTSCRIPT. The output includes physio-chemical properties, solvent accessibility, prediction reliability, and conservation number values for each amino acid. The consensus is decided by a simple majority of the six algorithms and if there is a tie, then the prediction from PHD is used. The consensus receives a Q_3 score of 72.9%, better than PHD's Q_3 score of 71.9%.

A small, but important, step in Q_3 score is achieved by using a consensus of six different algorithms on Cuff, Clamp, Siddiqui, Finlay, and Barton's server (1998). Though a 1% difference does not seem like much, when compared to how proteins can be comprised of over tens of thousands of amino acids it shows that this is a important accomplishment. What makes it even more important is that there is more than one path of improvement to this server in terms of attempting to improve its main algorithm (PHD), improving the five other algorithms, or even adding algorithms that follow different heuristics.

Cuff J. A., Clamp M. E., Siddiqui A. S., Finlay M. and Barton G. J. (1998). Jpred: A Consensus Secondary Structure Prediction Server. *Bioinformatics, 14*(10), 892-893.

Cuff and Barton (2000) state that training a neural network secondary structure prediction with different types of multiple sequence alignment profiles derived from the same sequences, is shown to have a top scoring accuracy of 76.4%. A scale from 0 to 9 is used as a confidence score of which a 5 or higher as having an average Q3 score of 84%. The improvements are from training with different representations of the same alignment data, which is described in detail.

Cuff and Barton (2000) state that methods for predicting secondary structure of proteins provide information that is useful both in *ab initio* prediction as well as additional constraints for threading algorithms. Different heuristics have been applied in *ab initio* prediction including: simple linear statistics, physio-chemical properties, linear discrimination, machine learning, neural networks, k-way, nearest neighbors, evolutionary trees, simple residue, substitution matrices, and combinations of the different methods. Neural network prediction methods give better results than the previous best consensus methods.

Cuff and Barton (2000) screened 513 proteins from their previous study in 1998 and removed any proteins that were shorter than 30 residues. A new testing set was made so that all of the algorithms could be tested equally on blind data rather than being limited to having strong results based on proteins that were in their training data. The SNNS neural network package was applied which allowed incorporation of the networks into a C function for stand-alone code. The network ensemble consisted of two artificial neural networks. The methods still included consensus combination, solvent accessibility prediction, and a confidence measure.

Cuff and Barton (2000) performed two types of testing: a seven-fold cross-validated test and a blind test on a new set of proteins. The blind test that was performed showed a 1.8% increase over their previous implementation of Jpred which had a 1% accuracy increase on PHD. This increase came in residues that had confidence scores below five. Previously residues with a confidence score below two had an average accuracy at or below 75%, but for the new Jnet, accuracy for these residues was higher than 76%.

The neural network algorithm used by Jnet for protein secondary structure prediction has increased the prediction accuracy, for the same sequences using the same basic algorithm, by 7%. The improvement changed the accuracy from 69.5% to 76.4%. In 6 years from 1993, secondary structure prediction accuracy has improved from 70.6% to over 76% with Jnet. Cuff and Barton (2000) state that with more recent work on genomics, including the human genome project, accuracies should improve even greater in the prediction of secondary structure of proteins.

Cuff, J. A., and Barton, G. J. (2000). Application of Enhanced Multiple Sequence Alignment Profiles to Improve Protein Secondary Structure Prediction. *PROTEINS: Structure, Function and Genetics 40,* 502-511.

Cole, Barber, and Barton (2008) mention the third revision of Jpred in which the secondary structure accuracy of is 81.5%. In the new Jpred 3 server, functionality has been added for significant usability improvements that give clearer feedback of the progress or failure of submitted requests. The functional improvements also include batch submission of sequences and the ability to send summaries via email. Updates to the search database were included as well in the functional improvements.

Cole, Barber, and Barton (2008) state that despite recent structural genomics initiatives, the gap between knowledge of protein structure and sequences continues to expand. There were less than 50,000 structures stored in the PDB and almost 5 million sequences in UniProt. Homology modeling is the current best approach for predicting secondary structure, but more than 50% proteins lack useable structural homologues. They further claim that, due to optimizations in the Jnet algorithm, accuracy went up from 76.4% to 81.5%, a 5.1% increase.

Cole, Barber, and Barton (2008) state that their server, Jpred, takes a single protein sequence or multiple sequence alignment and returns predictions made by the Jnet algorithm. The main differences, in the new Jnet algorithm, are the use of only PSI-BLAST Position-specific scoring matrix and HMMER Hidden Markov Model instead of frequency profiles. These improvements lead to around a 5% increase in structure prediction accuracy and roughly 2% increase in solvent accessibility prediction. The user interface of the Jpred website was been improved for better compliance and usage.

Cole, Barber, and Barton (2008) go on to state that secondary structure prediction is an important tool for the analysis of proteins, and that Jpred is one of those tools being well used as well as an accurate source. The recent update of Jpred include the Jnet v2.0 algorithm that improves secondary structure prediction to 81.5% and solvent accessibility predictions up to 88.9%. The most notable changes to Jpred have to do with user interaction by providing more feedback to users regarding progress or problems regarding their submissions.

The Jpred server optimizations, made by Cole, Barber, and Barton (2008), have kept Jpred at the top of the list of best secondary structure prediction tools. There are numerous tools for secondary structure prediction and many are accessible or available online. The Jpred server started by increasing accuracy of secondary structure prediction by 1%, the introduction of the Jnet algorithm the accuracy was raised by 3.1%, and optimizations to the Jnet algorithm have increased the secondary structure prediction accuracy another 5.1%. Further Jpred improvements and optimizations are eagerly awaited for.

Cole C., Barber J. D., and Barton G. J. (2008). The Jpred 3 secondary structure prediction server. *Nucleic Acids Research, 36*, W197–W201. doi:10.1093/nar/gkn238

The authors of Gapped BLAST and PSI-BLAST (1997) deals with the BLAST programs as being widely used tools for searching protein and DNA databases for similarities. Refinements have been done to the BLAST programs to enhance performance while decreasing the time it takes to execute. The refinements include a new criterion for the extension of word hits and a new heuristic for generating gapped alignments. Another method was added that uses significant alignments and combines them into a position-specific scoring matrix. The searching the database using the matrix produced a Position-Specific Iterated BLAST. Both run at three times the speed of the original.

Gapped BLAST and PSI-BLAST (1997) include information stating that BLAST is a heuristic that requires time proportional to the product of the lengths of the sequence and the database searched. The addition of several new algorithmic ideas allow new versions of BLAST to achieve better accuracy at a faster speed. Three important improvements were added to BLAST: The criterion for extending word pairs has been modified to a new ‘two-hit’ method; the addition of an algorithm for generating gapped alignments, and the production of a matrix to be used for a Position-Specific Iterated BLAST, or PSI-BLAST.

Altschul S. F., Madden T. L., Schäffer A. A., Zhang J., Zhang Z., Miller W., and Lipman D.J. (1997) began analyzing the BLAST algorithm by first looking at high scoring local alignments. The refined 'two-hit' algorithm that they produced is based on the observation that a high-scoring segment pair is longer than a single word pair, and therefore may have multiple hits on the same diagonal and within a short distance of each other. The PSI-BLAST program is usually more sensitive than the BLAST program, while being faster.

Gapped BLAST and PSI-BLAST (1997) were tested by first comparing protein family sequences through the SWISS-PROT database, and then running the matrices generated against a shuffled version of the same database. The resulting accuracy acknowledge that position-specific scoring matrices could be built automatically. The same eleven query sequences that were used above were also used to compare the speeds of Gapped BLAST and PSI-BLAST. For particular examples, PSI-BLAST took longer than Gapped BLAST, but for non-redundant databases PSI-BLAST performed exceptionally well.

BLAST has been widely used and is an excellent tool for multiple-sequence alignments and for finding homologous sequences. Improvements to the speed of BLAST make it be an even better option for incorporation into tools and algorithms. The BLAST programs are likely to see future improvements.

Altschul S. F., Madden T. L., Schäffer A. A., Zhang J., Zhang Z., Miller W., and Lipman D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research, 25*(17), 3389-3402.

Suzek B. E., Huang H., McGarvey P., Mazumder R., and Wu C. H. (2007) state that clustering of protein sequence space based on similarity helps organize them into manageable datasets and reduces over-representation. The UniRef database currently covers 44 million sequences with UniRef100 having identical sequences in a single UniRef entry. UniRef90 and UniRef50 are built by clustering UniRef100 sequences at the 90% and 50% identity levels. The database size reductions of UniRef100, UniRef90, and UniRef50 are 10%, 40%, and 70%. UniRef has been used in many research areas from genome annotation to proteomics analysis.

Suzek B. E., Huang H., McGarvey P., Mazumder R., and Wu C. H. (2007) go on to claim that clustering of similar sequences aids in identification of homologs as well as other analysis tasks and that a number of algorithms have been developed for clustering protein sequences. They have developed UniRef as one key component to have complete coverage of protein sequence space at 100%, 90%, and 50% identity. Many commonly used databases contain redundant sequences, which can include identical sequences of the same length. Representative databases can be used to provide the same amount of information with a smaller set of data.

The UniRef100 clusters are first generated by using sequences from UniProtKB and UniParc. The UniRef90 clusters are then generated by using UniRef100 clusters and UniRef50 clusters are generated using UniRef90 clusters. There is a three step process for generating clusters: first the CD-Hit algorithm is used to cluster sequences, then overlapping regions are checked for gapped alignments, and finally sequences less than eleven amino acids are checked. Creating the UniRef50 clusters at the 50% identity level is intensive, so a parallel clustering CD-Hit was created. (Suzek et al., 2007).

Suzek B. E., Huang H., McGarvey P., Mazumder R., and Wu C. H. (2007) state that the UniRef databases are generated and released every 2 weeks. They compare UniRef100 to the "nr protein" sequence database produced by NCBI claiming that it is similar in scope. UniRef still receives a 4% size reduction from "nr protein" databases. The releases of the UniRef databases are available online in XML and downloadable by FTP. PSI-BLAST and HMMER are examples of two groups that take advantage of the reduced sampling bias to develop and improve profile-based models.

The UniRef databases have been created to provide a complete coverage of the protein sequence space, with increased speed of searches and better detection of relationships by removing redundant sequences. The biweekly update that UniRef provides is an excellent tool as it provides up-to-date clusters which keeps pace with the rapid growth of the number of protein sequences. These major features allow research to flourish in both biological and computational settings and allows tools like HMMER and PSI-BLAST to be more efficient.

Suzek B. E., Huang H., McGarvey P., Mazumder R., and Wu C. H. (2007). UniRef: comprehensive and non-redundant UniProt reference clusters. *Bioinformatics, 23*(10), 1282–1288. doi:10.1093/bioinformatics/btm098

Eddy S. R. (1998) reviews the recent literature and software related to profile Hidden Markov Models methods. Profile HMMs turn a multiple sequence alignment into a position-specific scoring system that is suitable for searching databases for homology. Profile HMM analysis complements standard pairwise comparison methods for large sequence analysis. Hidden Markov Models performance is comparable to threading prediction models in the CASP2 exercise.

Eddy S. R. (1998) states that it seems desirable to use information from multiple-sequence alignments when searching data for homology and for this 'profile' methods they designed a way to build position-specific scoring models.

Eddy S. R. (1998) explains that are various types of HMMs and that it is best to first understand the main HMM theory before considering special cases. The name 'hidden markov model' comes from first-order Markov chain, but with only the symbol sequence being directly observed. Once a HMM is drawn, there are dynamic programming algorithms for aligning and scoring sequences. HMMs have been applied to gene finding, radiation hybrid mapping, genetic linkage mapping, phylogenetic analysis, and protein secondary structure prediction.

Eddy S. R. (1998) claims that several packages implement profile HMMs which include HMMER, and others. HMMs have been found useful in fold recognition protein structure prediction. Many fold recognition methods are not called HMMs, but are called sequence threading algorithms; however, any threading scoring system that uses dynamic programming as a solution can be remade into a probabilistic Hidden Markov Model.

Eddy S. R. (1998) states that in 1998 the human genome project threatened to overwhelm the analysis in sequence data and goes on to state that HMMs were a hope for being able to analyze all of the raw sequence data. HMMs could provide a second tier of analysis that would complement BLAST and FASTA analysis.

Eddy S. R. (1998). Profile hidden Markov models. *Bioinformatics, 14*(9), 755-763.

Simons K. T., Bonneau R., Ruczinski I., and Baker D. (1999) introduced an *ab initio* approach at CASP3. They designed a Monte Carlo simulated annealing procedure for predicting tertiary structures of proteins which involves testing random simlar structures for lower energy against the current lowest energy structure. The scoring function used in this procedure consists of sequence independent terms representing several different measurement terms. For each of 21 small, *ab initio* protein targets, 1,200 resulting structures were constructed, each took 100,000 attempted fragment substitutions. The results were encouraging and they suggest that *ab initio* methods could soon become useful for simple structure predictions for proteins that lack a close homologue.

Simons K. T., Bonneau R., Ruczinski I., and Baker D. (1999) claim that the motivation for their approach to protein tertiary structure prediction is that local interactions bias segments of the chain to sample distinct sets of local structures. Local structures distributions in known three-dimensional structures are used as an approximation to the distribution of structures sampled by isolated peptides. The primary non-local interactions that are considered are hydrophobic burial, electrostatics, disulfide bonding, main chain hydrogen bonding, and excluded volume. The model makes *ab initio* structure prediction of small proteins computationally feasible.

Simons K. T., Bonneau R., Ruczinski I., and Baker D. (1999) begin with a multiple sequence alignment generated by PSI-BLAST using default parameters and omit sequences with less than 25% or more than 90% sequence identity to a target. Structure prediction for sequences related to proteins of known structure was not attempted. The edited multiple sequence alignments were sent to the PHD which predicted secondary structure of each three- and nine-residue segments. Then protein tertiary structures were generated from these sets of fragments using the insertion-simulated annealing.

Simons K. T., Bonneau R., Ruczinski I., and Baker D. (1999) predicted the structures of 21 out of the 43 targets that were given in CASP3 that did not have obvious homologues. Eighteen determined structures of those twenty-one are available for comparison to the predicted structures. When the structures were made available, the predictions were evaluated by searching for substructures. After the CASP3 meeting, predictions made by others provided a reference point for evaluating the designed method.

Simons K. T., Bonneau R., Ruczinski I., and Baker D. (1999) conclude with four points: Low-resolution structure prediction can succeed without explicit representation, human intervention can be bias, it is unclear how much multiple sequence information contributed to their predictions, and that the method, with additional information, can potentially be useful for threading/homology modeling in cases of very low sequence similarity.

Simons K. T., Bonneau R., Ruczinski I., and Baker D. (1999). Ab Initio Protein Structure Prediction of CASP III Targets Using ROSETTA. *PROTEINS: Structure, Function, and Genetics, 37*(3), 171-176. doi: 10.1002/(SICI)1097-0134(1999)37:3+<171::AID-PROT21>3.0.CO;2-Z

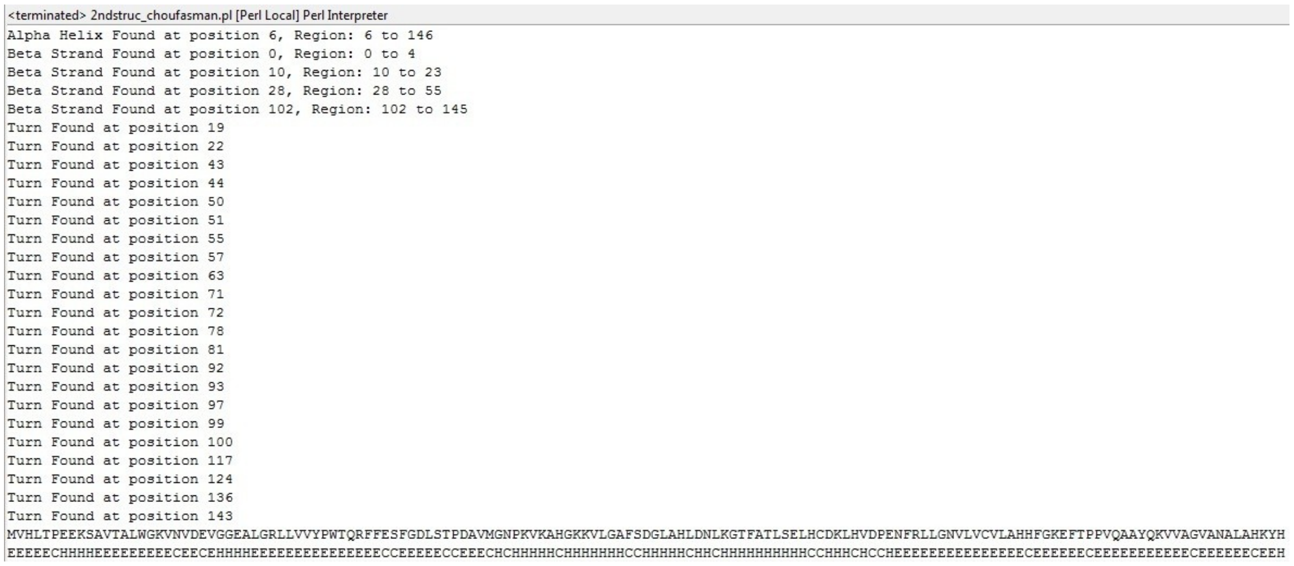
There are many more papers and texts on the subject of protein structure prediction with the preceding being only a small number of them specifically chosen due to their importance to this work. Only secondary structure prediction is focused in this work as going further becomes increasingly more complex. In the next section, we discuss the design of several methods used in predicting secondary structure including the use of amino acid probabilities, Hidden Markov Models, and neighboring amino acids.

III. Design

i. Initial

First, the Chou-Fasman Secondary Structure Prediction method was implemented. The algorithm is from the 1970s and the average prediction success of the algorithm on a protein is 50%-60%. *Exploring Bioinformatics* (2010) has an excellent description and overview of the algorithm with a partial implementation. The implemented part is the prediction of α-helices and the student is required to implement the prediction of β-sheets, β-turns, and overlaps. The implementation of the algorithm was not extremely difficult and was accomplished in Perl. The book states that there are many ways to handle overlaps and gives some hints as to what to try, but is not explicit in the best way to proceed. Instead it states the choice of how to handle overlaps are what distinguish different Chou-Fasman implementations.

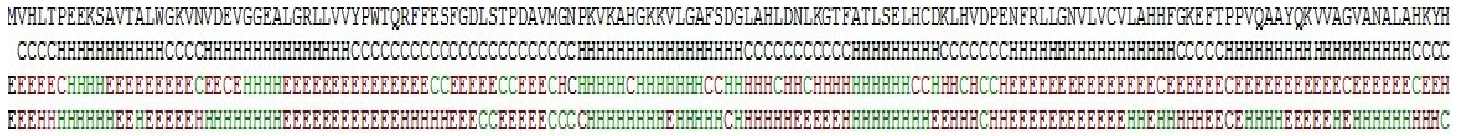
The structure of HBB has been known for some time and is interesting in that there are no β-sheets, making HBB an excellent choice to use for on initial tests. Since there is only one structural element in HBB testing it in allows a deeper look of how the algorithm performs toward that element. The exact HBB sequence used can be found in Fasta format of >gi|455997|gb| AAA16334.1| beta-globin [Homo sapiens].



```
<terminated> 2ndstruc_choufasman.pl [Perl Local] Perl Interpreter
Alpha Helix Found at position 6, Region: 6 to 146
Beta Strand Found at position 0, Region: 0 to 4
Beta Strand Found at position 10, Region: 10 to 23
Beta Strand Found at position 28, Region: 28 to 55
Beta Strand Found at position 102, Region: 102 to 145
Turn Found at position 19
Turn Found at position 22
Turn Found at position 43
Turn Found at position 44
Turn Found at position 50
Turn Found at position 51
Turn Found at position 55
Turn Found at position 57
Turn Found at position 63
Turn Found at position 71
Turn Found at position 72
Turn Found at position 78
Turn Found at position 81
Turn Found at position 92
Turn Found at position 93
Turn Found at position 97
Turn Found at position 99
Turn Found at position 100
Turn Found at position 117
Turn Found at position 124
Turn Found at position 136
Turn Found at position 143
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
EEEEECHHHHEEEEEEEEEEECEECEHHHHEEEEEEEEEEEEEEEEEEECCEEEEEECCEEECHCHHHHHCHHHHHHHCCHHHHHCHHCHHHHHHHHHHHCCHHHCHCCHEEEEEEEEEEEEEEEEEEEECEEEEEECEEEEEEEEEEEECEEEEEEECEEH
```

Figure 3. Implemented ChouFasman Method on HBB. *Exploring Bioinformatics* guided project implementation of the Chou-Fasman method with types of regions found and consensus on bottom.

Figure 3 shows the actual prediction of the implemented Chou-Fasman method as described by *Exploring Bioinformatics* (2010). The first problem, with the Chou-Fasman method is that it overestimates the amount of β-sheets in the secondary structure of a protein. Since HBB does not have any β-sheets, any number of predicted β-sheets is incorrect and the number of possible β-sheets that are predicted includes over half of the protein. It is also notable that the predicted α-helix region is almost the entire protein. The implementation of the Chou-Fasman algorithm seemed reasonable when tested against other proteins, but did not do well when tested against HBB since it over-estimates β-sheets. The Chou & Fasman Secondary Structure Prediction Server(CFSSP*)*, which uses an implementation of the Cho-Fasman algorithm, was located online and was used as a comparison tool to the implemented method that was described in the book.



```
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
 CCCCHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHCCCCCCCCCCCHHHHHHHHHCCCCCCCHHHHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHHHHHHHHHHHCCCC
EEEEECHHHHEEEEEEEEEECEECEHHHHEEEEEEEEEEEEEEEECCEEEEECCEEECHCHHHHHCHHHHHHHCCHHHHHCHHCHHHHHHHHHHCCHHHCHCCHEEEEEEEEEEEEEEEEEECEEEEEECEEEEEEEEEEEECEEEEEEECEEH
EEEHHHHHHHHEEHEEEEEHHHHHHHHHEEEEEEEEEEEEEEHHHHHEEECCEEEEEECCCCHHHHHHHHHEHHHHHCHHHHHHEEEEEHHHHHHHHHHEEHHHCHHEEEEEEEEEEEEEHHEHHHHHEECEHHHHEEEEEHEHHHHHHHHHC
```

Figure 4. Prediction Results of HBB in Implemented Method and CFSSP. The lists of sequences in the figure is as follows: the primary structure of HBB, the secondary structure of HBB, the implemented secondary structure prediction of HBB, and CFSSP secondary structure prediction of HBB. Red is used for incorrect predicted structure and green for correctly predicted. H represents an α-heliex, E represents a β-sheet, and C represents a random coil.

Figure 4 shows the secondary structure predictions of the implemented method and of CFSSP. A structure comparison tool was written and showed that the implemented Chou-Fasman method achieved a Q_3 score of 29% with 42 out of 146 residues predicted correctly while CFSSP performed better with a Q_3 score of 40% predicting 59 out of 146 residues correctly. *Exploring Bioinformatics* (2010) mentions that there are many different ways to implement the Chou-Fasman Method and it is my belief that the reason for the discrepancy between the scores is because of different implementations. For instance, in one test on another protein, CFSSP predicted a region of α-helices that spanned only two proteins. The implementation described in *Exploring Bioinformatic*s (2010) does not allow for a region of less than 5 proteins to be an α-helix region. CFSSP seems to handle overlaps better than the method described in *Exploring Bioinformatics* (2010). Both tools did not give high-confidence predicitions of HBB structure and both predicted a large amount of β-sheet regions where there should be none so a more refined prediction tool was needed.

Since more than one method would be implemented, the next step was to make or look for a training set and a testing set which would then allow a standard for comparison. To generate these sets, a tool from NCBI called Cn3d was used. This tool displays protein structures in three-dimensional views. The file that Cn3d uses comes compressed, but if saved from the program it reverts into a readable text format. A lot of information is available in the Cn3d structure files, and a program was written to extract the secondary structure of the protein that is stored in the file.

ExPASy Proteomics Server was used as another starting point in locating tools for comparison, a training set, and a testing set. Since HBB performed so poorly on the ChouFasman method, it was used again for testing on the tools that were listed and working on the ExPASy site, eleven tools in all. The tools performed remarkably well and their results are listed in Figure 5.

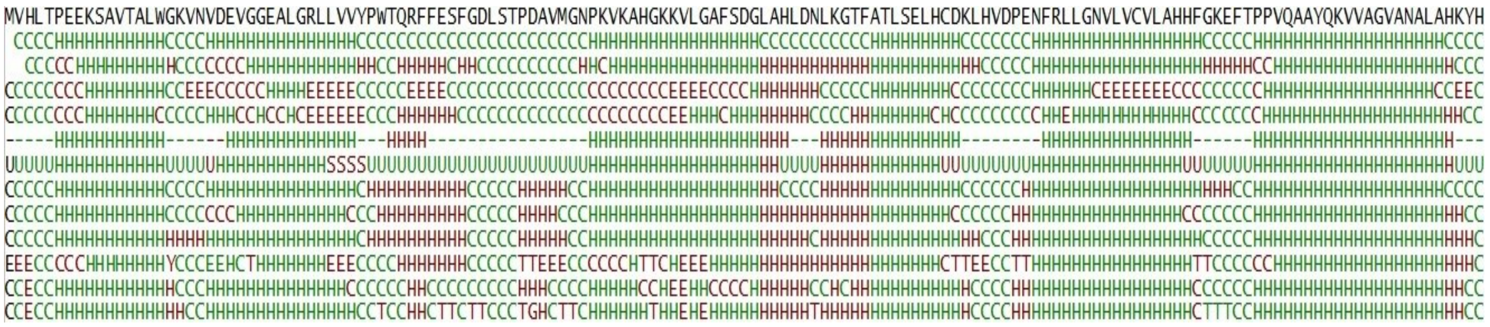


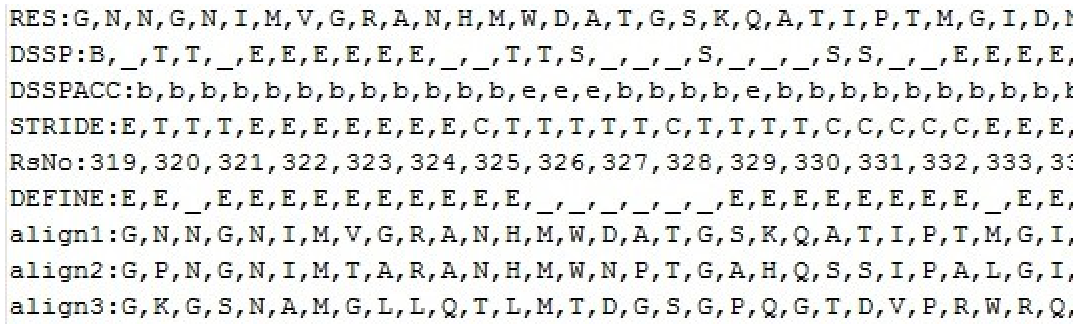
Figure 5. Prediction Results of HBB Used On Eleven Tools Listed on ExPASy website. Starting from the third line, The following tools predictions of HBB are shown from top to bottom: APSSP, GOR4, HNN, Jpred, Jufo, Porter, Prof, PSIPred, SOPMA, SSPRO, and SSPRO(8 class). The first line is the primary structure of HBB and the second line is the secondary structure. Again, red stands for incorrectly predicted structure for residues and green for correctly predicted. α-helices are represented by H, β-sheets by E, and random coils by C, U, and dash.

Figure 5 shows the performance of eleven tools that are linked on the ExPASy Proteomics Server under secondary structure prediction. The first sequence listed on the figure is the primary structure of HBB and the second line is the known secondary structure that each tool was compared against. The lines following the second are the predicted secondary structures by different tools. The accuracy and tools from top to bottom are as follows: APSSP2 with 73%, GOR4 with 60%, HNN with 68%, Jpred with 88%, Jufo with 88%, Porter with 82%, Prof with 76%, PsiPred with 75%, SOPMA with 59%, SSPro with 0.79%, and SSPro(8) with 0.74%. Several of these tools compare to PHD, which uses similar sequences in position scoring. Jpred and Jufo scored the highest with both having 129 out of 146 residues predicted correctly. The next highest was Porter with 82% accuracy which is 6% lower than Jpred and Jufo's. SOPMA and GOR4 scored the lowest at around 60%. The two predicted structures from Jpred and Jufo were 90% alike, with different areas guessed incorrectly. Jpred was chosen as a tool to compare against, since it performed well and the web server hosting Jpred provided the training set and testing set that was used for Jnet1 algorithm.

Unfortunately, the training and testing sets that are available on the Jpred server are for Jpred2, not the current version, Jpred3. A quick overview of the algorithms used in Jpred1-3 is as follows: Jpred1 used a consensus-based approach between top tools; Jpred2 introduced the Jnet algorithm which involved a position-specific scoring matrix(PSSM) and a neural network model; Jpred3 introduces Jnet2 which is an optimized version of the previous algorithm and sets. Jpred1 achieved an average Q3 score of 72.9%, Jpred2 had an average Q3 score of 76.4%., Jpred3, which has been optimized, had an even higher average Q3 score at 81.5%. The performance of Jpred is excellent compared to other available secondary structure prediction tools, and it is a good choice to use.

Jpred is a three step neural network approach to structure prediction currently running Jnet v2.0. The first step in the analysis of an amino acid sequence is an optional BLAST query against sequences in the PDB using a 0.0005 e-value cut-off. This step is only used determine if a similar protein exists and is not used for secondary structure prediction. The next step is a search against the UniRef90 database that provides clustered sets of sequences with 90% similarity. The third step is to produce a Hidden Markov Model(HMM) profile using the HMMer program and a Position-specific Scoring Matrix(PSSM) profile from PSI-BLAST. The PSSM generation has three iterations where an e-value of 0.05 is used as a cut-off for the first iteration and then an e-value of 0.01 is used for the subsequent two iterations. The final step also involves inputting the HMM and the PSSM profiles into Jnet which produces secondary structure predictions as well as solvent accessibility predictions, which relate to how much area of each amino acid is on the outside structure of the protein. Since two of the steps involve the program looking for similar sequences, it seemed reasonable to attempt to take the HMM aspect of Jpred and use it for secondary structure prediction. Jpred uses profile HMMs which rely on similiar sequences to build predictions for secondary structures since similiar sequences would have similiar structures.

The Jpred3 team has named the training set CB513 and the testing set CB406. The names of the sets are based on the number of proteins in the set. This is the data that they used to train and test Jnet1 for Jpred2 which achieved a final Q_3 accuracy of 76.4%. The Jpred3 team states that both the CB513 and CB406 data sets are non-redundant.



```
RES:G,N,N,G,N,I,M,V,G,R,A,N,H,M,W,D,A,T,G,S,K,Q,A,T,I,P,T,M,G,I,D,
DSSP:B,_,T,T,_,E,E,E,E,E,E,_,_,T,T,S,_,_,_,S,_,_,_,S,S,_,_,E,E,E,E,
DSSPACC:b,b,b,b,b,b,b,b,b,b,b,b,e,e,e,b,b,b,b,e,b,b,b,b,b,b,b,b,b,
STRIDE:E,T,T,T,E,E,E,E,E,E,E,C,T,T,T,T,T,C,T,T,T,T,C,C,C,C,C,E,E,E,
RsNo:319,320,321,322,323,324,325,326,327,328,329,330,331,332,333,3
DEFINE:E,E,_,E,E,E,E,E,E,E,E,E,E,_,_,_,_,_,E,E,E,E,E,E,E,E,_,E,E,
align1:G,N,N,G,N,I,M,V,G,R,A,N,H,M,W,D,A,T,G,S,K,Q,A,T,I,P,T,M,G,I,
align2:G,P,N,G,N,I,M,T,A,R,A,N,H,M,W,N,P,T,G,A,H,Q,S,S,I,P,A,L,G,I,
align3:G,K,G,S,N,A,M,G,L,L,Q,T,L,M,T,D,G,S,G,P,Q,G,T,D,V,P,R,W,R,Q,
```

Figure 6. CB513 Training Set File for Jnet1. This is file 3cox-2-AS.1.all that came in the CB513 training set. The files in the training set contain 9 lines that all include the residues in the sequence and the secondary structure of the sequence as well as other information.

Figure 6 shows the contents of a representative file in the CB513 training data. The information in the file is as follows: each residue in the protein, protein structure by DSSP, solvent accessibility, protein structure by knowledge-based STRIDE*,* residue number, protein structure definition, and then three alignments. Since the project is to be designed for an *ab initio* approach, only the residues and the secondary structure were used for training.

The CB406 testing set has a similar structure to that of the training set, but instead of just having 9 lines of information, there are 84 lines of information. This is because the CB406 testing set comes with predictions of the protein using the tools DSC, NNSSP, PBLOCK, PHD, Predator, Zpred, Jpred, Jnet, as well as any extra information that is produced by the tools. The tools may have been optimized since the release of the available CB406 testing set, so they may perform better than listed in the files. Both sets are excellent to have and are available online on the Jpred server. The results achieved, as stated by Cole, Barber, and Barton (2008), on the CB406 test set is 76.4% accuracy with Jnet version 1 and 81.5% with Jnet version 2. When there is only a single similar sequence available for the prediction, the accuracy of Jnet drops to 65.9%.

After acquiring training and testing sets, the next step was to generate a novel algorithm. The goal is to focus on different obtainable information, which may involve using different algorithms without resorting to using similar sequences. For using an *ab initio* approach, I decided that it was worth trying a Hidden Markov Model(HMM). HMMs have been used successfully in speech recognition as well as in protein structure prediction. (Eddy, 1998) They do not get results that are as good as neural networks in protein secondary structure prediction. With the Jpred training set, a simple HMM matrix was designed and implemented in Perl.

Matrices of values were created which included counts of how many times an amino acid appears as a type of secondary structure, counts of transitions between secondary structures, and initial starting secondary structures. These matrices were then converted into probability matrices and further into log odds. They were then used in HMMs with the Viterbi Algorithm when given a sequences of amino acids as input to produce a prediction of the secondary structure of the amino acid sequence. Since Jpred uses homologs in producing their HMM, this approach attempts to eliminate that need.

ii. First-order Markov Chain HMM

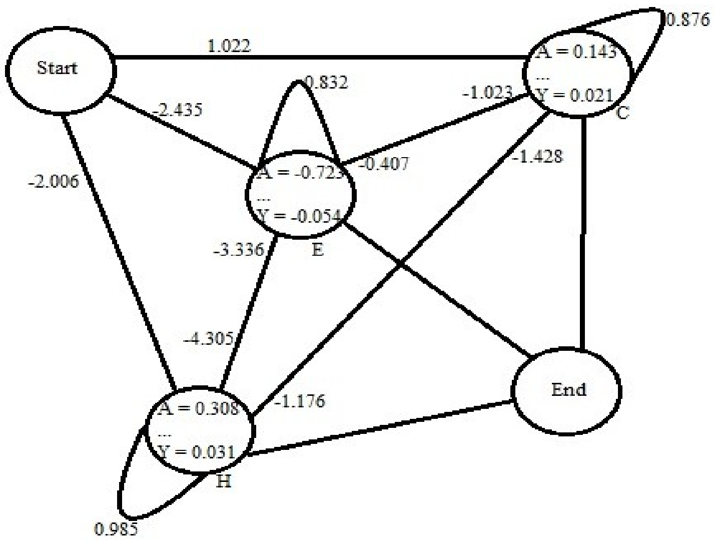


Figure 7. HMM with Three States. Here is a HMM with the three secondary structures that an amino acid may take determined by log-odds. Initial, transitional, and amino acid log-odds for A and Y are displayed. The states contain log-odds for all twenty amino acids, but only A and Y are shown.

The first HMM designed, which is displayed in Figure 7, has three states that are for α-helix (H), β-sheet (E), and random coil (C). Within each state are probabilities converted to log-odds of each amino acid taking that state's particular secondary structure. The lines between states are transitional probabilities, which are also converted to log-odds with lines coming from the Start having initial probabilities to each secondary structure. The HMM was also used in a model without transitional and initial probabilities, so that a probabilistic view of the sequence per amino acid could be seen as an attempt to be able to obtain additional information from each amino acid.

iii. Second-order Markov Chain HMM

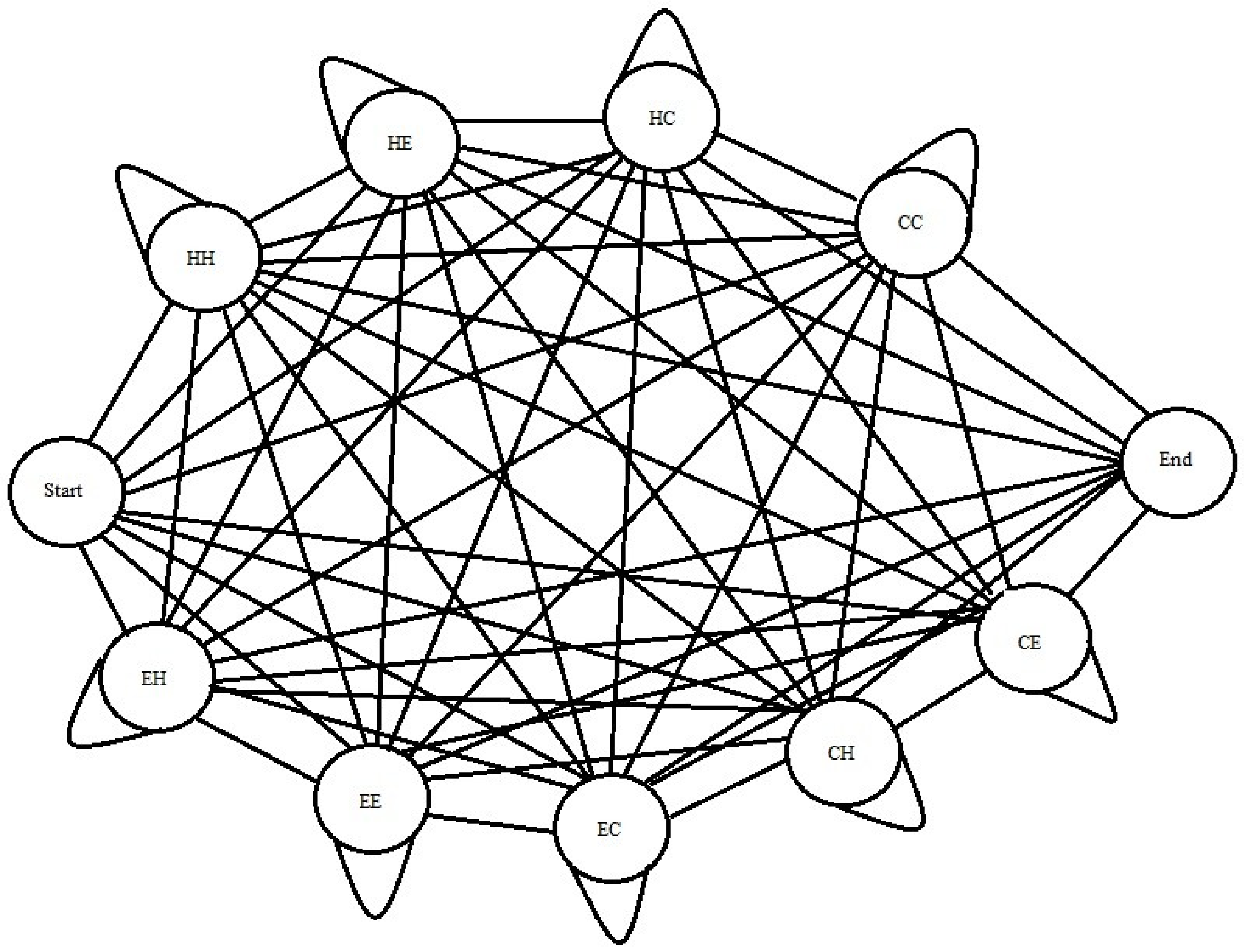


Figure 8. HMM with Nine States. Here is a more complex HMM that is comparable to a second-order Markov Chain, but is displayed as a single-order Markov Chain with nine states. This HMM has pairs of amino acids in states to determine secondary structure.

Figure 8 describes the second HMM that was designed. This one instead takes the previous amino acid into account with the current amino acid. The HMM was used in two methods: first, taking pairs of amino acids and assigning them a state by highest probability, and second, limiting transitions between states. In this second method, the model would only step forward one amino acid, so if the initial chosen state was EH, the only available transitions would be HH, HE, and HC. This HMM's purpose was to attempt to find a relationship between an amino acid, its secondary structure, and the previous amino acid. Just moving from the first to second model exponentially increased the size of the HMM. It is possible to expand HMMs further, but it is not practical to do so.

The size of a matrix involving pairs of amino acids is 400 by 9, a reasonable amount of data to work with would need to fill the 3600 positions many times over. Using sets of three amino acids instead of two increases the matrix size up to 8000 by 27, which has 216,000 positions. Using a set of four amino acids would bring the matrix size up to 160000 by 81, or 12,960,000 positions. Because of the enormous size of the data sets, I had to abandon the HMM approach beyond the second-order.

Two other methods involving probabilities were attempted. The first being incorporating the start and stop probabilities. The idea seemed very reliable, but the application of the idea was not easy. The method was an expansion of the initial HMM, which had initial, transitional, and secondary structure probabilities. The transitional probabilities were changed to favor amino acids that have higher probabilities of being a certain structure. Having the probabilities for the above mentioned values was not enough to accurately determine secondary structure and required more information, so another method was attempted.

iv. Amino Acid Neighbor Method

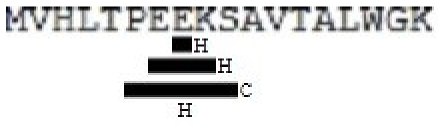


Figure 9. Working with an Amino Acids Neighbors. This was the fourth method attempted where the prediction of secondary structure is based on probabilities on nearby amino acids in the sequence.

Figure 9 shows the fourth type of method that was attempted. It is essentially a variation of nearest neighbor: the probabilities of amino acids surrounding the chosen amino acids are used in predicting the secondary structure of that amino acid. Other methods use windows with sizes of fifteen or more positions, but are used for sequence similarity to predetermined structures. The goal of this method was to find how accurately nearby amino acids could be used in helping deteremine a protein's structure. Through testing, it became apparent that the four surrounding amino acids seemed to play the biggest role in an amino acid becoming a certain secondary structure with the preceding two having the biggest role. With more optimization, this method could be added to some tools to improve the ability of the tool to predict secondary structure. Since the current implementation of the method works best for α-helix regions, it might be best to take the route that the designers of Jpred1 took and use a consensus of several different methods or have specialized implementations for β-sheets and possibly coil regions. The main point of the current implementation was to show possible areas for improvement to current tools that are available. In the next section, implementations of the discussed methods are described as well as how to run them on a Windows computer.

IV. Implementation of a*b initio* Learning Methods

i. Overview

There were several programs implemented for the purpose of secondary structure prediction. All implementations were done in the Perl language and, to be able to run the programs, Perl is required as well as the specified input files for the particular program. The specified input files can either be located online or created. Perl was chosen as the programming language since it is excellent for sequence manipulation and as a scripting language there was no need to constantly compile the files to run. The downside is that, the only way to run the files is to have a Perl interpreter installed. The instructions to be able to run the programs are listed below, followed by a brief description of each program.

ii. Program Files

2ndstruc_choufasman.pl

- **Description**: This file is a version of the ChouFasman method with which I tried to optimize. I also tried different implementations than the ones given in *Exploring Bioinformatics*. The file still uses the ChouFasman method from the book as a starting point.
- **Input**: Takes 2 files as input.
  i. *ch7ex1in.txt* – This file contains the sequence in Fasta format.
  ii. *choufasman.txt* – This file contains the 7 ChouFasman parameters in a matrix for each of the 20 amino acids.
- **Output**: Outputs some information to the console as well as to a file.
  i. Console – Outputted is every region found that is a a-helix, b-sheet, or turn. A single secondary structure sequence is also displayed.
  ii. *ch7ex1out.txt* - A single secondary structure sequence is outputted to this file.

2ndstruc_choufasman2.pl

- **Description**: This file is a version of the ChouFasman method that is exactly described by *Exploring Bioinformatics.* The file was later modified in an attempt and optimize and saved as *2ndstruc_choufasman.pl*
- **Input**: Takes 2 files as input.
  i. ch7ex1in.txt – This file contains the sequence in Fasta format.
  ii. choufasman.txt – This file contains the 7 ChouFasman parameters in a matrix for each of the 20 amino acids.
- **Output**: Outputs some information to the console as well as to a file.
  i. Console – Outputted to the console is every region found that is a a-helix, b-sheet or turn.
  ii. ch7ex1out.txt - A single secondary structure sequence is outputted to this file.

2ndstruc_cmp.pl

- **Description**: This program compares 2 sequences and prints out how similiar the 2 sequences are.
- **Input**: The program asks for input from the console.
  - **i.** Console – The console asks for the first sequence and then for a second sequence.
- **Output**: The program outputs to the console.
  - **i.** Console – The number of matching characters, the length of the query, and the percentage similarity are outputted.

2ndstruc_makematrix.pl

- **Description**: This program creates matrices involving amino acids, secondary structure, transitions between secondary structure, initial probability, and a probability matrix.
- **Input**: The program needs the CB513 training set and a second file which is the list of files in the directory of the training set.
  - **i.** 513_distribute/ – The Jpred2 CB513 training set which can be download from http://www.compbio.dundee.ac.uk/www-jpred/about.html.
  - **ii.** 513_distribute/list.txt – Text file with the list of all the files located in the training set directory excluding the list.txt file. This can be created by navigating to the directory in command line and typing *dir /B >list.txt*
- **Output**: The program has 6 different output files which are used for different programs.
  - **i.** 1matrix.txt – A 20x3 matrix of amino acids that counts the number of times each amino acid appears and the type of secondary structure, a-helix, b-sheet, or random coil using the DSSP values in the CB513 training set.
  - **ii.** 1matrix2.txt – A 20x3 matrix of amino acids that counts the number of times each amino acid appears and the type of secondary structure, a-helix, b-sheet, or random coil using the DEFINE values in the CB513 training set.
  - **iii.** 1imatrix.txt – A 1x3 matrix of count of each type of secondary structure that begins a protein.
  - **iv.** 1tmatrix.txt – A 3x3 matrix of the counts that one secondary structure transitions into another.
  - **v.** 1t20matrix.txt – A 20x60 matrix of each amino acid's count of moving from one amino acid into another with specific counts of each type of secondary structure that the transitioning amino acid is.
  - **vi.** 1pmatrix.txt – The 1matrix.txt converted into probability values.

2ndstruc_make2xmatrix.pl

- **Description**: This program creates matrices involving pairs of amino acids and secondary structure. The matrices are again saved as counts which can be later converted to other formats.
- **Input**: The program needs the CB513 training set as well as another file which is the list of files in the directory of the training set.
  - **i.** 513_distribute/ – The Jpred2 CB513 training set which can be download from http://www.compbio.dundee.ac.uk/www-jpred/about.html.
  - **ii.** 513_distribute/list.txt – Text file with the list of all the files located in the training set

directory excluding the list.txt file. This can be created by navigating to the directory in command line and typing *dir /B >list.txt*

- **Output**: The program has 2 different output files.
  - **i.** 2matrix.txt – A 400x9 matrix of amino acids that counts the number of times each amino acid appears and the type of secondary structure, α-helix, β-sheet, or random coil using the DSSP values in the CB513 training set.
  - **ii.** 2matrix2.txt – A 400x9 matrix of amino acids that counts the number of times each amino acid appears and the type of secondary structure, α-helix, β-sheet, or random coil using the DEFINE values in the CB513 training set.

2ndstruc_1aa_transitions.pl

- **Description**: The program takes matrices built by 2ndstruc_makematrix.pl and converts them into log-odds and uses these values to predict against HBB or the CB513 training set.
- **Input**: The program needs the CB513 training set as well as the matrices that are built in another program as well as the list of files in the directory of the training set.
  - **i.** 513_distribute/ – The Jpred2 CB513 training set which can be download from http://www.compbio.dundee.ac.uk/www-jpred/about.html.
  - **ii.** 513_distribute/list.txt – Text file with the list of all the files located in the training set directory excluding the list.txt file. This can be created by navigating to the directory in command line and typing *dir /B >list.txt*
  - **iii.** 1matrix.txt – A 20x3 matrix from 2ndstruc_makematrix.pl or p1matrix.txt
  - **iv.** 1matrix2.txt – A 20x3 matrix from 2ndstruc_makematrix.pl or p1matrix2.txt
  - **v.** 1imatrix.txt – A 1x3 matrix from 2ndstruc_makematrix.pl or p1imatrix.txt
  - **vi.** 1tmatrix.txt – A 3x3 matrix from 2ndstruc_makematrix.pl or p1tmatrix.txt
  - **vii.** 1t20matrix.txt – A 20x60 matrix from 2ndstruc_makematrix.pl or p1t20matrix.txt
- **Output**: The program outputs to the console as well as outputting log-odd matrices if not done previously.
  - **i.** Console – The number of correctly predicted residues, the total number of residues, and the average percentage that the program gets correct.
  - **ii.** p1matrix.txt – A 20x3 matrix of log-odds created from the 1matrix.txt if does not exist.
  - **iii.** p1matrix2.txt – A 20x3 matrix of log-odds created from the 1matrix2.txt if does not exist.
  - **iv.** p1imatrix.txt – A 1x3 matrix of log-odds created from the 1imatrix.txt if does not exist.
  - **v.** p1tmatrix.txt – A 3x3 matrix of log-odds created from the 1tmatrix.txt if does not exist.
  - **vi.** p1t20matrix.txt – A 20x60 matrix of log-odds created from the 1t20matrix.txt if does not exist.

2ndstruc_1aa.pl

- **Description**: The program takes matrices built by 2ndstruc_makematrix.pl and converts them into log-odds and uses these values to predict against HBB, the CB513 training set, or the CB406 testing set. The difference between this program and the previous one is that this program does not take into account initial or transition probabilities.
- **Input**: The program needs the CB513 training set as well as the matrices that are built in another program as well as the list of files in the directory of the training set.

  - **i.** 513_distribute/ – The Jpred2 CB513 training set which can be download from http://www.compbio.dundee.ac.uk/www-jpred/about.html.
  - **ii.** 513_distribute/list.txt – Text file with the list of all the files located in the training set directory excluding the list.txt file. This can be created by navigating to the directory in command line and typing *dir /B >list.txt*
  - **iii.** 1matrix.txt – A 20x3 matrix from 2ndstruc_makematrix.pl or p1matrix.txt
  - **iv.** 1matrix2.txt – A 20x3 matrix from 2ndstruc_makematrix.pl or p1matrix2.txt
- **Output**: The program outputs to the console as well as outputting log-odd matrices if not done previously.
  - **i.** Console – The number of correctly predicted residues, the total number of residues, and the average percentage that the program gets correct.
  - **ii.** p1matrix.txt – A 20x3 matrix of log-odds created from the 1matrix.txt if does not exist.
  - **iii.** p1matrix2.txt – A 20x3 matrix of log-odds created from the 1matrix2.txt if does not exist.

2ndstruc_2aa.pl

- **Description**: The program takes matrices built by 2ndstruc_make2xmatrix.pl and converts them into log-odds and uses these values to predict against HBB, the CB513 training set, or the CB406 testing set. The prediction made is based on sets of two amino acids.
- **Input**: The program needs 2matrix.txt produced by 2ndstruc_make2xmatrix.pl and asks for input from the console.
  - **i.** 2matrix.txt – A 400x9 matrix from 2ndstruc_make2xmatrix.pl or p2matrix.txt
- **Output**: The program outputs to the console as well as outputting a log-odd matrix if it has not been made yet.
  - **i.** Console – The number of matching characters, the total number of residues, and the percentage similarity are outputted.
  - **ii.** p2matrix.txt – A 400x9 matrix of log-odds created from the 2matrix.txt if does not exist.

2ndstruc_2aaby1.pl

- **Description**: The program takes matrices built by 2ndstruc_make2xmatrix.pl and converts them into log-odds and uses these values to predict against HBB, the CB513 training set, or the CB406 testing set. The prediction made is based on sets of two amino acids, taking in account the previous predictions.
- **Input**: The program needs 2matrix.txt produced by 2ndstruc_make2xmatrix.pl and asks for input from the console.
  - **i.** 2matrix.txt – A 400x9 matrix from 2ndstruc_make2xmatrix.pl or p2matrix.txt
- **Output**: The program outputs to the console as well as outputting a log-odd matrix if it has not been made yet..
  - **i.** Console – The number of matching characters, the total number of residues, and the percentage similarity are outputted.
  - **ii.** p2matrix.txt – A 400x9 matrix of log-odds created from the 2matrix.txt if does not exist.

2ndstruc_motifmake.pl

- **Description**: This program creates matrices involving amino acid and secondary structure based on sub sequences of amino acid and creates a matrix file as output.
- **Input**: The program needs the CB513 training set as well as another file which is the list of files in the directory of the training set.
  - **i.** 513_distribute/ – The Jpred2 CB513 training set which can be download from http://www.compbio.dundee.ac.uk/www-jpred/about.html.
  - **ii.** 513_distribute/list.txt – Text file with the list of all the files located in the training set directory excluding the list.txt file. This can be created by navigating to the directory in command line and typing *dir /B >list.txt*
- **Output**: The program outputs the matrix that is made.
  - **i.** 5motif.txt – A 20x5 matrix of amino acids that assigns a positive number to α-helix, a negative number to β-sheets, and zero to random coils. Then based on the amino acids position in sub sequences, calculates the number of times each amino acid appears as what type of secondary structure, α-helix, β-sheet, or random coil using the DSSP values in the CB513 training set.

2ndstruc_motiftest.pl

- **Description**: The program takes the probability matrix, 1pmatrix.txt, and uses it as a special case of testing involving taking lengths of amino acids to predict the current amino acid.
- **Input**: The program needs 1pmatrix.txt produced by 2ndstruc_makematrix.pl and asks for input from the console.
  - **i.** 1pmatrix.txt – A 20x3 matrix of probability values of 1matrix.txt
- **Output**: The program outputs to the console.
  - **i.** Console – The number of matching characters, the total number of residues, and the percentage similarity are outputted.

2ndstruc_makematrix_start_stop.pl

- **Description**: This program creates matrices involving amino acid and secondary structure based on wheather the amino acid is starting, ending, or in the middle of the specific secondary structure.
- **Input**: The program needs the CB513 training set as well as the directory of the training set.
  - **i.** 513_distribute/ – The Jpred2 CB513 training set which can be downloaded from http://www.compbio.dundee.ac.uk/www-jpred/about.html.
  - **ii.** 513_distribute/list.txt – Text file with the list of all the files located in the training set directory excluding list.txt. This can be created by navigating to the directory in command line and typing *dir /B >list.txt*
- **Output**: The program outputs two matrices that are created.
  - **i.** 1sssmatrix.txt – The 20x9 matrix of counts of the number of times an amino acid starts, is in, and ends each secondary structure type.
  - **ii.** 1zzzmatrix.txt – 20x9 probability matrix of the number of times an amino acid starts, is in, and ends each secondary structure type.

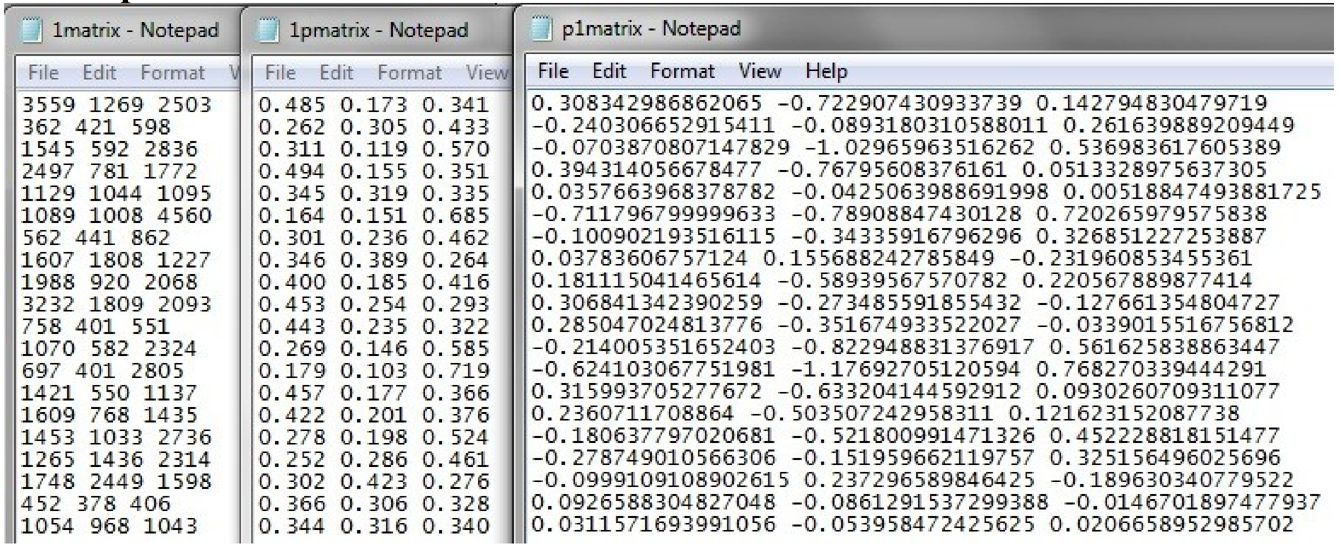
2ndstruc_probtest.pl

- **Description**: The program takes the probability matrices 1pmatrix.txt and 1zzzmatrix.txt and uses it as a special case of testing involving the probabilities of amino acids starting, stopping, and being in a secondary structure.
- **Input**: The program needs 1pmatrix.txt produced by 2ndstruc_makematrix.pl and 1zzzmatrix.txt produced by 2ndstruc_makematrix_start_stop.pl.
  i. 1pmatrix.txt – A 20x3 matrix of probability values of 1matrix.txt.
  ii. 1zzzmatrix.txt – A 20x9 matrix of start, in, stop, probabilities values.
- **Output**: The program outputs to the console.
  i. Console – The number of matching characters, the total number of residues, and the percentage similarity are outputted.

2ndstruc_count_cb406.pl

- **Description**: The program takes the CB406 test set and counts the number of times each secondary structure appears without taking anything else into account.
- **Input**: The program needs the CB406 test set.
  i. 406_distribute/ – The Jpred2 CB513 training set which can be downloaded from http://www.compbio.dundee.ac.uk/www-jpred/about.html.
  ii. 406_distribute/list.txt – Text file with the list of all the files located in the training set directory excluding the list.txt file. This can be created by navigating to the directory in command line and typing *dir /B >list.txt*
- **Output**: The program outputs to the console.
  i. Console – The number of times each secondary structure appears in the test set.

iv. Sample Matrix Files



```
3559 1269 2503
362 421 598
1545 592 2836
2497 781 1772
1129 1044 1095
1089 1008 4560
562 441 862
1607 1808 1227
1988 920 2068
3232 1809 2093
758 401 551
1070 582 2324
697 401 2805
1421 550 1137
1609 768 1435
1453 1033 2736
1265 1436 2314
1748 2449 1598
452 378 406
1054 968 1043
```

```
0.485 0.173 0.341
0.262 0.305 0.433
0.311 0.119 0.570
0.494 0.155 0.351
0.345 0.319 0.335
0.164 0.151 0.685
0.301 0.236 0.462
0.346 0.389 0.264
0.400 0.185 0.416
0.453 0.254 0.293
0.443 0.235 0.322
0.269 0.146 0.585
0.179 0.103 0.719
0.457 0.177 0.366
0.422 0.201 0.376
0.278 0.198 0.524
0.252 0.286 0.461
0.302 0.423 0.276
0.366 0.306 0.328
0.344 0.316 0.340
```

```
0.308342986862065 -0.722907430933739 0.142794830479719
-0.240306652915411 -0.0893180310588011 0.261639889209449
-0.0703870807147829 -1.02965963516262 0.536983617605389
0.394314056678477 -0.76795608376161 0.0513328975637305
0.0357663968378782 -0.0425063988691998 0.00518847493881725
-0.711796799999633 -0.78908847430128 0.720265979575838
-0.100902193516115 -0.34335916796296 0.326851227253887
0.0378360675712 4 0.155688242785849 -0.231960853455361
0.18111504146561 4 -0.58939567570782 0.220567889877414
0.306841342390259 -0.273485591855432 -0.127661354804727
0.285047024813776 -0.351674933522027 -0.0339015516756812
-0.214005351652403 -0.822948831376917 0.561625838863447
-0.624103067751981 -1.17692705120594 0.768270339444291
0.315993705277672 -0.633204144592912 0.0930260709311077
0.2360711708864 -0.503507242958311 0.12162315208773 8
-0.180637797020681 -0.521800991471326 0.452228818151477
-0.278749010566306 -0.151959662119757 0.325156496025696
-0.0999109108902615 0.237296589846425 -0.189630340779522
0.0926588304827048 -0.0861291537299388 -0.0146701897477937
0.0311571693991056 -0.053958472425625 0.020665895298570 2
```

Figure 10. Sample Matrices. Here are three files that are made using the CB513 training set. The left matrix is the counts that each amino acid appears as each secondary structure. The middle matrix is the probability values and the right matrix is the log-odds values.

Figure 10 shows the matrices created by 2ndstruc_makematrix.pl when used with the CB513 training set. The left most column is the probability of the amino acid being part of an α-helix, middle column for β-sheet, and right column for random coil. The amino acids go in alphabetical order based on their single letter code from top to bottom. With little alteration, other sets can be used to create matrices that could be used with the programs. In the next chapter, we present the various results obtained from running the implemented methods on HBB, the CB513 training set, and the CB406 testing set.

V. Results

i. Overview

Some of the current top prediction tools use large window sizes that range up to seventeen amino acids long or take position specific scoring matrices created by matching similar sequences. Studies show that a window size of up to fifteen positions for α-helices and nine positions for β-sheets could be used for optimal prediction. (Chen et al. 2006) To have more accuracy at specific amino acids, a smaller window size may be better while a large window size may work better for larger proteins or similar sequences. A larger window size may account for interactions between two different regions of secondary structures within a protein, while a smaller one may more accurately detect the affect of an amino acid's neighbors. The main result = presented in this paper is that there are data available in sequences that need to be found and exploited. Not enough is understood in the subject of protein structure prediction. More data analysis and testing need to be done to discover relationships between amino acids, the primary structure, and the secondary structure. Notable results that are produced by the method that uses relations between amino acids are shown in Figure 11.

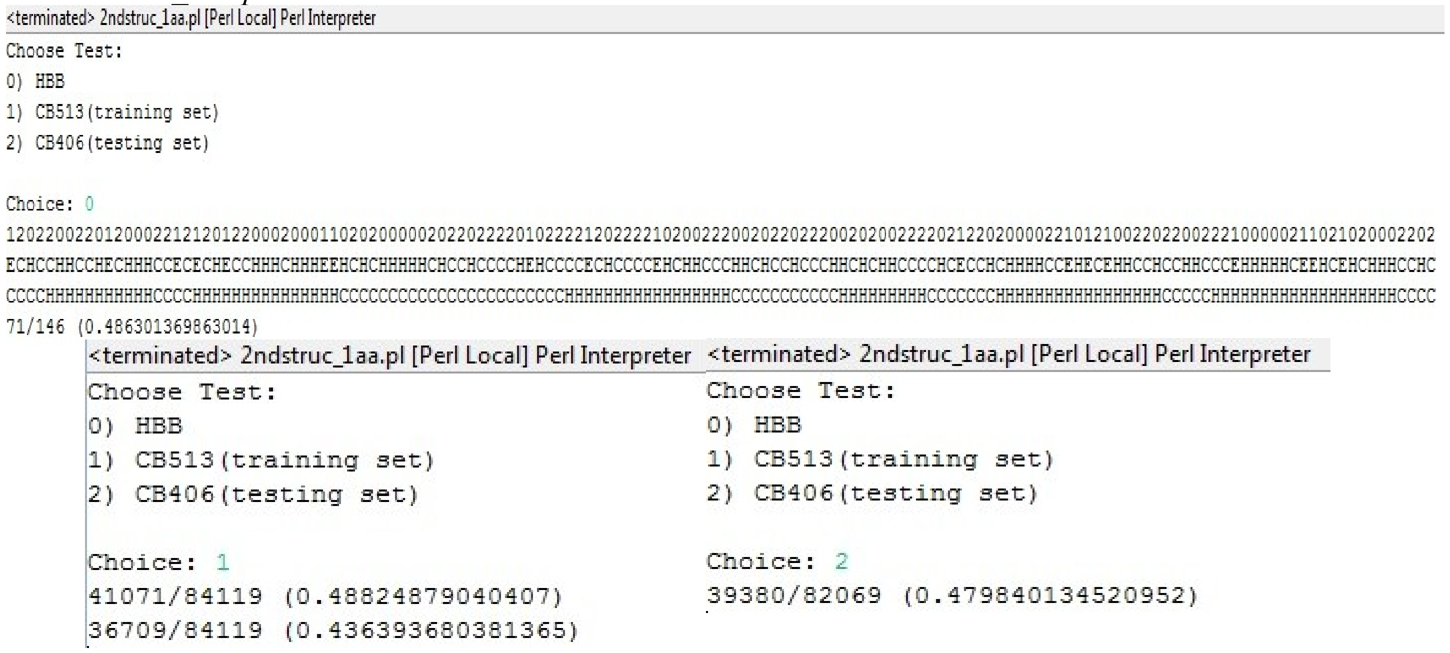
	Nearest Neighbor Implementation	**Jpred**	**GOR**	**SOPMA**
HBB	64.38%	88.00%	60.00%	59.00%
CB513 training set 1bpha-1-DOMAK.all	85.71%	80.95%	38.10%	47.62%
CB513 training set 1coi-1-AS.all	96.55%	96.55%	68.97%	86.21%
CB513 training set 1gcmc-1-AUTO.1.all	93.94%	87.88%	69.70%	84.85%
CB406 testing set 1gg2g.concise	94.44%	92.59%	70.37%	77.78%

Figure 11. Main Table of Improved Predictions. The above table shows the prediction of the implemented nearest amino acid method against Jpred, GOR, and SOMPA. The table list prediction accuracy for HBB, three proteins from the CB513 training set, and one protein from the CB406 set that the implemented methods performs better on.

Figure 11 shows the results of the nearest neighbor implemented methods against Jpred, GOR, and SOPMA on HBB and four proteins in the training and testing sets. On the four proteins available in the sets, the method preforms as well as Jpred or better. On 1coi-1-AS.all protein, the implemented method performs as well as Jpred, but predicts a different area of results which is worth mention because a combination of both methods may produce a superior result. In the next section, we show the individual runs that are performed by the different prediction methods, as well as the runs of the nearest neighbor method that performs better than some other available tools.

ii. Individual Runs

2ndstruc_1aa.pl



```
<terminated> 2ndstruc_1aa.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 0
1202200220120002212120122000200011020200000202202222010222212022221020022200202202220020200222202122020000221012100220220022210000021102102000022202
ECHCCHHCCHECHHHCCECECHECCHHHCHHHEEHCHCHHHHHCHCCHCCCCHEHCCCCECHCCCCEHCHHCCCHHCHCCHCCCHHCHCHHCCCCHCECCHCHHHHCCEHECEHHCCHCCHHCCCEHHHHHCEEHCEHCHHHCCHC
CCCCHHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHCCCCCCCCCCCHHHHHHHHHCCCCCCHHHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHHHHHHHHHHHHCCCC
71/146 (0.486301369863014)
```

```
<terminated> 2ndstruc_1aa.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 1
41071/84119 (0.48824879040407)
36709/84119 (0.436393680381365)
```

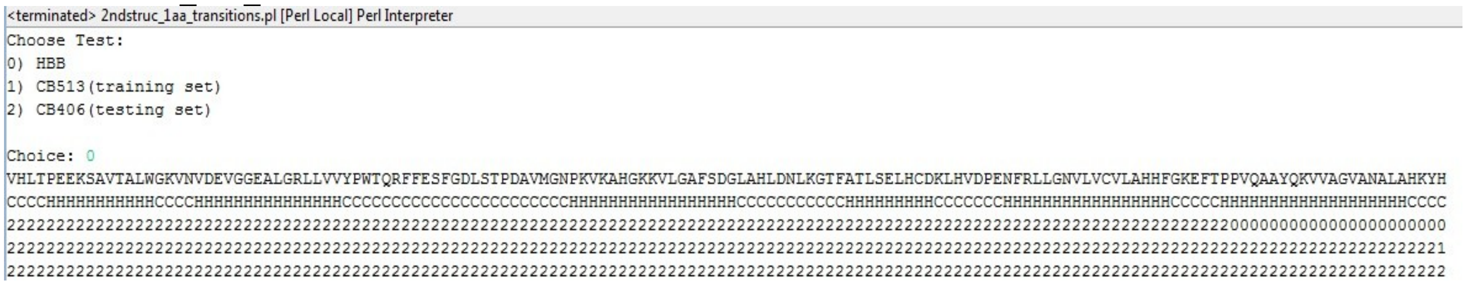
```
<terminated> 2ndstruc_1aa.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 2
39380/82069 (0.479840134520952)
```

Figure 12. Three Runs of 2ndstruc_1aa. The figure shows the runs of 2ndstruc_1aa, which only uses amino acid probability for secondary structure, on HBB, the CB513 training set, and the CB406 testing set.

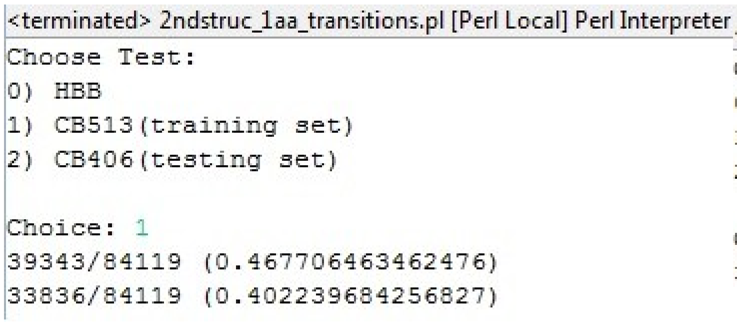
Figure 12 shows the three different runs for the 2ndstruc_1aa.pl program which only uses amino acid's probability for secondary structure type. Another way to state the program is that it is a HMM with only state probabilities and no initial or transitional probabilities. The program was not expected to perform well. Cole C., Barber J. D., and Barton G. J. (2008) state that an accuracy of randomly guessing the secondary structure of a protein should average around 42%. This program with only using a training set and not looking for any relationship when predicting a protein's secondary structure does slightly better than that. The program is 6% above the random average that should be received. It is most likely the higher probability proteins like Proline(P) ,which has a 72% chance of being part of a coil, that this program emphasizes.

The CB513 training set test has two probabilities listed as results with the top one measuring the predicted secondary structure against the DSSP and the bottom against DEFINE in the training set files. The DEFINE secondary structure is used in the training set as having another test since the methods were trained on the DSSP secondary structure given. Another important point, is that the program performs just as well on the training set as it does on the testing set which should be expected of an *ab initio* method. This was one of the first implemented methods along with the HMM that included initial and transitional probabilities whose results are described next.

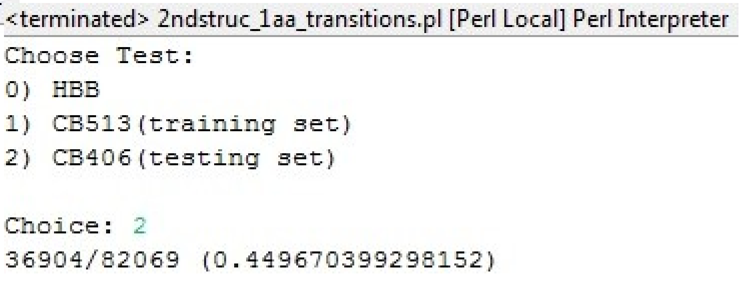
2ndstruc_1aa_transitions



```
<terminated> 2ndstruc_1aa_transitions.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 0
VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
CCCCHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHCCCCCCCCCCCHHHHHHHHHCCCCCCHHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHHHHHHHHHCCCC
22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222220000000000000000000000
22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222221
22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
```



```
<terminated> 2ndstruc_1aa_transitions.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 1
39343/84119 (0.467706463462476)
33836/84119 (0.402239684256827)
```



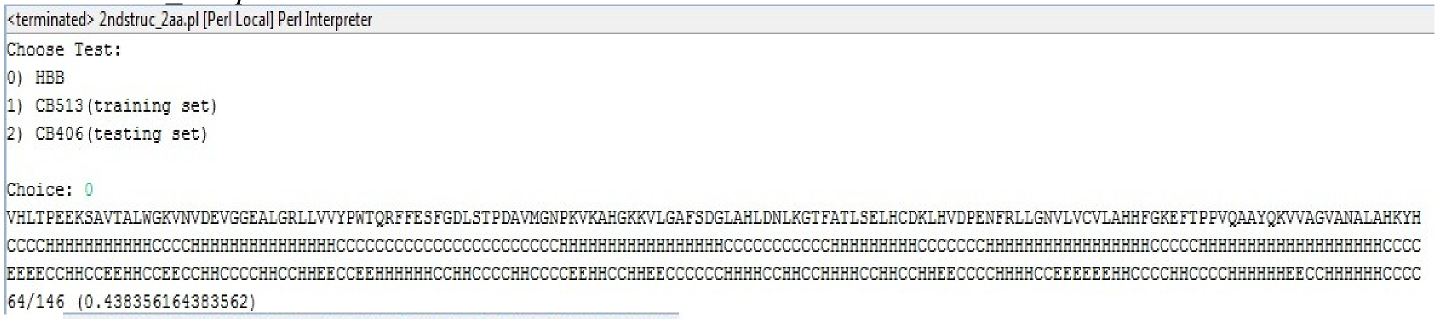
```
<terminated> 2ndstruc_1aa_transitions.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 2
36904/82069 (0.449670399298152)
```

Figure 13. Three Runs of 2ndstruc_1aa_transitions. The above figure shows the runs of 2ndstruc_1aa_transitions, which is the program which uses initial probability, transitional probability, and probability for secondary structure, on HBB, the CB513 training set, and the CB406 testing set.

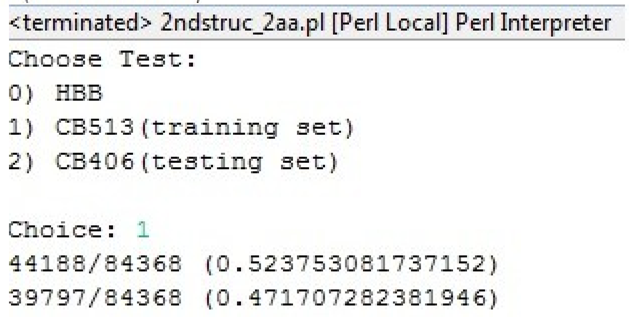
Figure 13 shows the the runs of 2ndstruc_1aa_transitions, which incorporates a HMM with initial probabilities, transitional probabilities and state probabilities of amino acids to secondary structure, on HBB, CB513 training set, and CB406 testing set. The Viterbi Algorithm was used in finding the optimal secondary structure for an amino acid sequence to take based on the HMM values. As seen from the run on HBB, the three final paths of the Viterbi Algorithm are shown. Unfortunately, the probability for amino acids to stay in the same secondary structure ends up much higher than to transition to another structure. This method performed worse than the previous method that did not use initial or transitional probabilities.

This method would not be useful to use. It did lead to the idea to have a different target of 0% amino acids predicted correctly. Being able to achieve this number would then be able to incorporate the method to remove one-third of the choices each amino acid can take since with 0% accuracy the method would guarantee the chosen secondary structure to not be the correct one for each amino acid. The next described results are for the implementation that was designed using the second-order Markov Chain HMM approach.

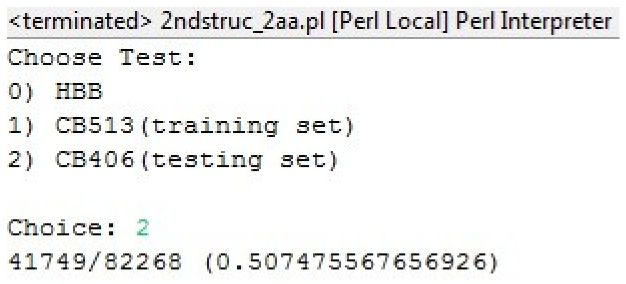
2ndstruc_2aa.pl



```
<terminated> 2ndstruc_2aa.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 0
VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
CCCCHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHCCCCCCCCCCHHHHHHHHHHCCCCCCCHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHHHHHHHHHCCCC
EEEECCHHCCEEHHCCEECCHHCCCCHHCCHHEECCEEHHHHHHCCHHCCCCHHCCCCEEHHCCHHEECCCCCCHHHHCCHHCCHHHHCCHHCCHHEECCCCCHHHHCCEEEEEEHHCCCCHHCCCHHHHHHEECCHHHHHHCCCC
64/146 (0.438356164383562)
```



```
<terminated> 2ndstruc_2aa.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 1
44188/84368 (0.523753081737152)
39797/84368 (0.471707282381946)
```



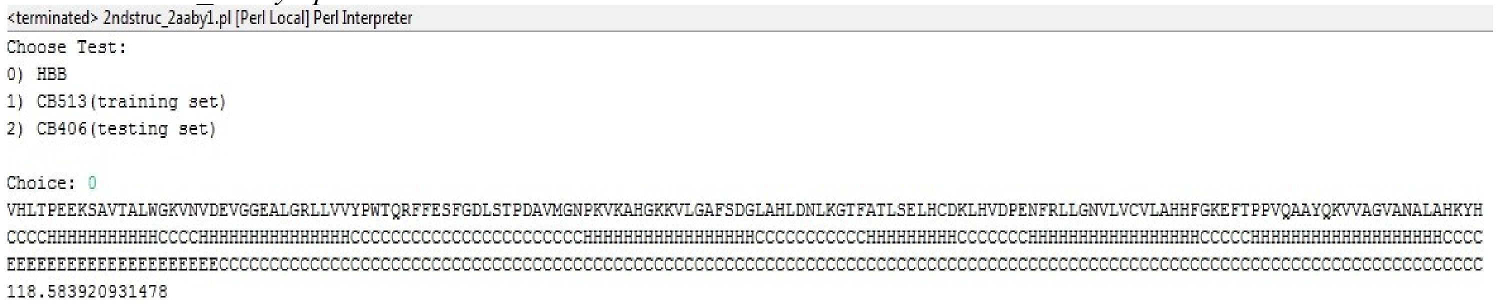
```
<terminated> 2ndstruc_2aa.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 2
41749/82268 (0.507475567656926)
```

Figure 14. Three Runs of 2ndstruc_2aa. The figure shows the runs of 2ndstruc_2aa which uses the probability pairs of amino acids for secondary structure, on HBB, the CB513 training set, and the CB406 testing set.

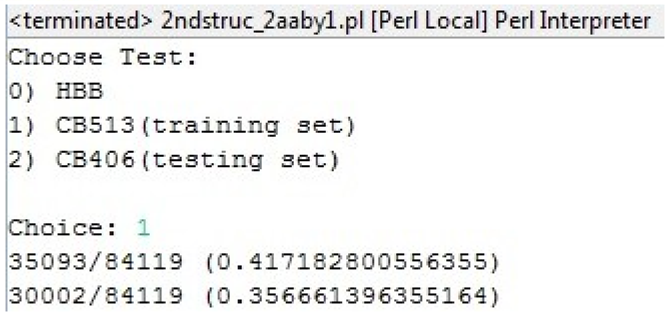
Figure 14 shows the runs of 2ndstruc_2aa which uses the probabilities taken from the CB513 training set where the program only uses pairs of amino acids incrementing to the next pair of amino acids for the next prediction. On the HBB run, the test does not do remarkably well. Again, as with 2ndstruc_1aa, the program performs nearly the same against the training set and the testing set. This, combined with the fact that the program does roughly 4% better than the 2ndstruc_1aa.pl program, shows that the secondary structure in which an amino acid participates is affected by that its neighbors.

The program performed better than the ChouFasman method on HBB. Alone, the algorithm may not perform very well, but it does show that grouping amino acids provides a better structure prediction than looking at them individually. The key is understanding the strength of each amino acid's contribution to nearby amino acids' secondary structures. The next results show the amino acid pair matrix probabilities used with single steps rather than moving to the next pair.

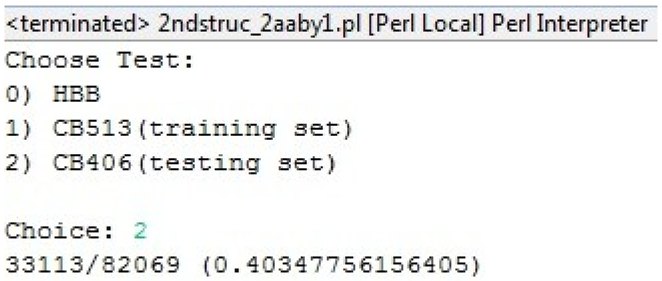
2ndstruc_2aaby1.pl



```
<terminated> 2ndstruc_2aaby1.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 0
VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
CCCCHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHCCCCCCCCCCCHHHHHHHHHCCCCCCCHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHHHHHHHHHCCCC
EEEEEEEEEEEEEEEEEEEEECCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
118.583920931478
```



```
<terminated> 2ndstruc_2aaby1.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 1
35093/84119 (0.417182800556355)
30002/84119 (0.356661396355164)
```



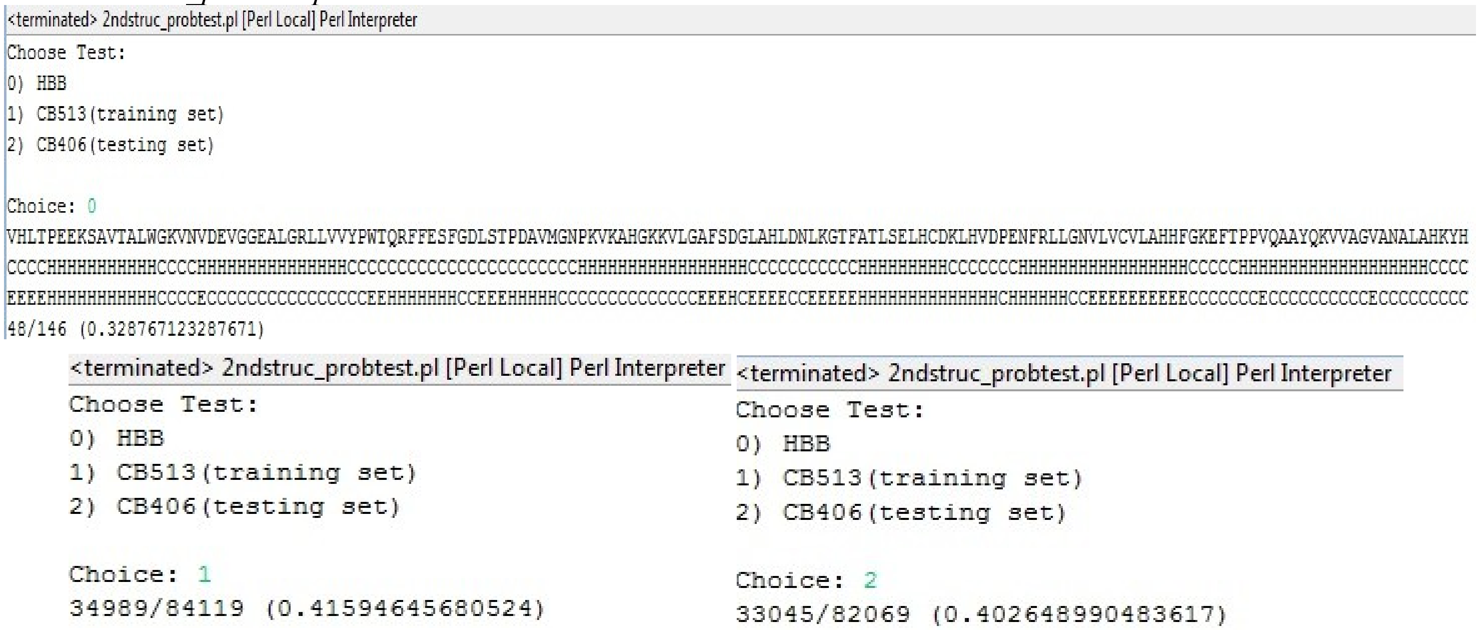
```
<terminated> 2ndstruc_2aaby1.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 2
33113/82069 (0.40347756156405)
```

Figure 15. Three Runs of 2ndstruc_2aaby1. The figure shows the runs of 2ndstruc_2aaby1 which uses the probability pairs of amino acids and increments amino acids by single steps to take previously predicted structure on HBB, the CB513 training set, and the CB406 testing set.

The program performs worse than a random algorithm for predicting secondary structure because it suffers from the same problem that the 2ndstruc_1aa_transitions.pl method does. Amino acids are more likely to stay in the same secondary structure than to change to other secondary structures. The program limits the number of state transitions from nine choices to three choices since it takes the previously guessed structure into account for the next prediction. For each continuing run, if a state EC is the highest probable secondary structure the next amino acid would have three choices H, E, and C while taking into account that the previous amino acid was C. This would mean that the three states that are tested are CH, CE, and CC.

The poor performance of this program alone makes it completely unfeasible to use. Transitional probabilities seem to yet again be the problem leaving long stretches of secondary structure with little to no transition. The program reinforced the idea that future work of *ab initio* could be an approach where a 0% accuracy secondary structure is predicted. Since an amino acid takes only 1 out of 3 secondary structures, targeting an incorrect structure has a higher probability since there are 2 out of 3 secondary structures that the amino acid does not take. In an attempt to improve transitional probabilities, the program brought up the idea of expanding transitional probabilities of each amino acid to look for amino acids that would be likely to start a new secondary structure or be on the end.

2ndstruc_probtest.pl



```
<terminated> 2ndstruc_probtest.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 0
VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
CCCCHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHCCCCCCCCCCCHHHHHHHHHCCCCCCCHHHHHHHHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHHHHHHCCCC
EEEEHHHHHHHHHHHCCCCECCCCCCCCCCCCCCCCEEHHHHHHHCCEEEHHHHHCCCCCCCCCCCCCCEEEHCEEEECCEEEEEHHHHHHHHHHHHHHCHHHHHHCCEEEEEEEEEECCCCCCCECCCCCCCCCCECCCCCCCCC
48/146 (0.328767123287671)
```

```
<terminated> 2ndstruc_probtest.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 1
34989/84119 (0.41594645680524)
```

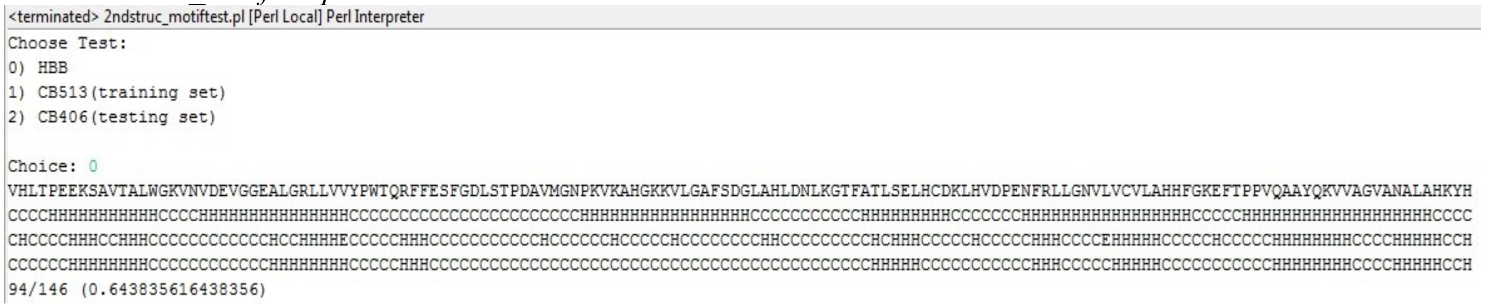
```
<terminated> 2ndstruc_probtest.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 2
33045/82069 (0.402648990483617)
```

Figure 16. Three Runs of 2ndstruc_probtest. The figure shows the runs of 2ndstruc_probtest which uses amino acid's probabilities to start, stop, and participate in a secondary structure. The runs are on HBB, the CB513 training set, and the CB406 testing set.

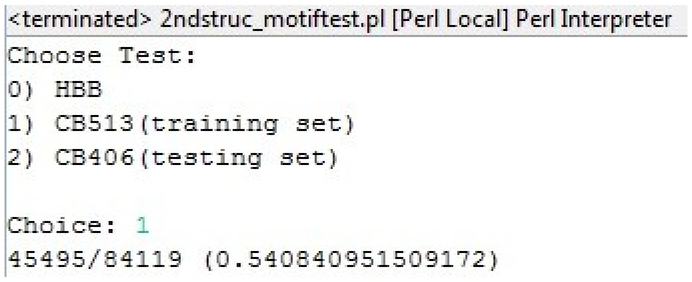
Figure 16 shows the third attempt at trying to use probability with secondary structure transitions. The third attempt did equally as poorly as the other two attempts which were 2ndstruc_1aa_transitions and 2ndstruc_2aaby1, but the HBB shows that the method tried to transition several times more. This third failed attempt at transitional probabilities does not mean that using probabilities relating to secondary structure transitions does not work, but instead needs to be focused more directly toward sub-regions and where amino acids would start or stop secondary structure regions. 2ndstruc_probtest uses probabilities of amino acids starting each secondary structure, being part of a secondary structure, and stopping a secondary structure. When traversing the sequence, three checks are being made whether or not a new secondary structure is starting or not.

The runs on the CB513 training set and CB406 testing set are under the 42% random probability which means that there is no guarantee that this method would have any success. The third attempt is the most promising with taking into account transitions, but is still not useable for protein structure prediction. The next implementation takes into account an amino acid's probabilities for secondary structures as well as probabilities of the amino acid's neighbor.

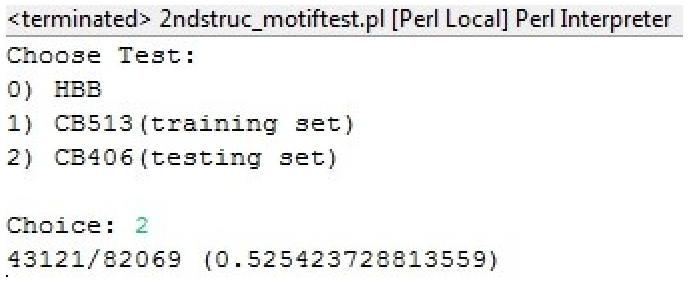
2ndstruc_motiftest.pl



```
<terminated> 2ndstruc_motiftest.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 0
VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
CCCCHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHHCCCCCCCCCCHHHHHHHHHCCCCCCHHHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHHHHHHHHHHHCCCC
CHCCCCHHHCCHHHCCCCCCCCCCCCCHCCHHHHECCCCCHHHCCCCCCCCCCCCHCCCCCHCCCCCHCCCCCCCHHCCCCCCCCCHCHHHCCCCCHCCCCCHHHCCCCEHHHHHCCCCCHCCCCCHHHHHHHHCCCCHHHHHHCCH
CCCCCCHHHHHHHHCCCCCCCCCCCCHHHHHHHHCCCCHHHCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCHHHHHCCCCCCCCCCCHHHCCCCCHHHHHHCCCCCCCCCCCHHHHHHHHHCCCCHHHHHHCCH
94/146 (0.643835616438356)
```



```
<terminated> 2ndstruc_motiftest.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 1
45495/84119 (0.540840951509172)
```

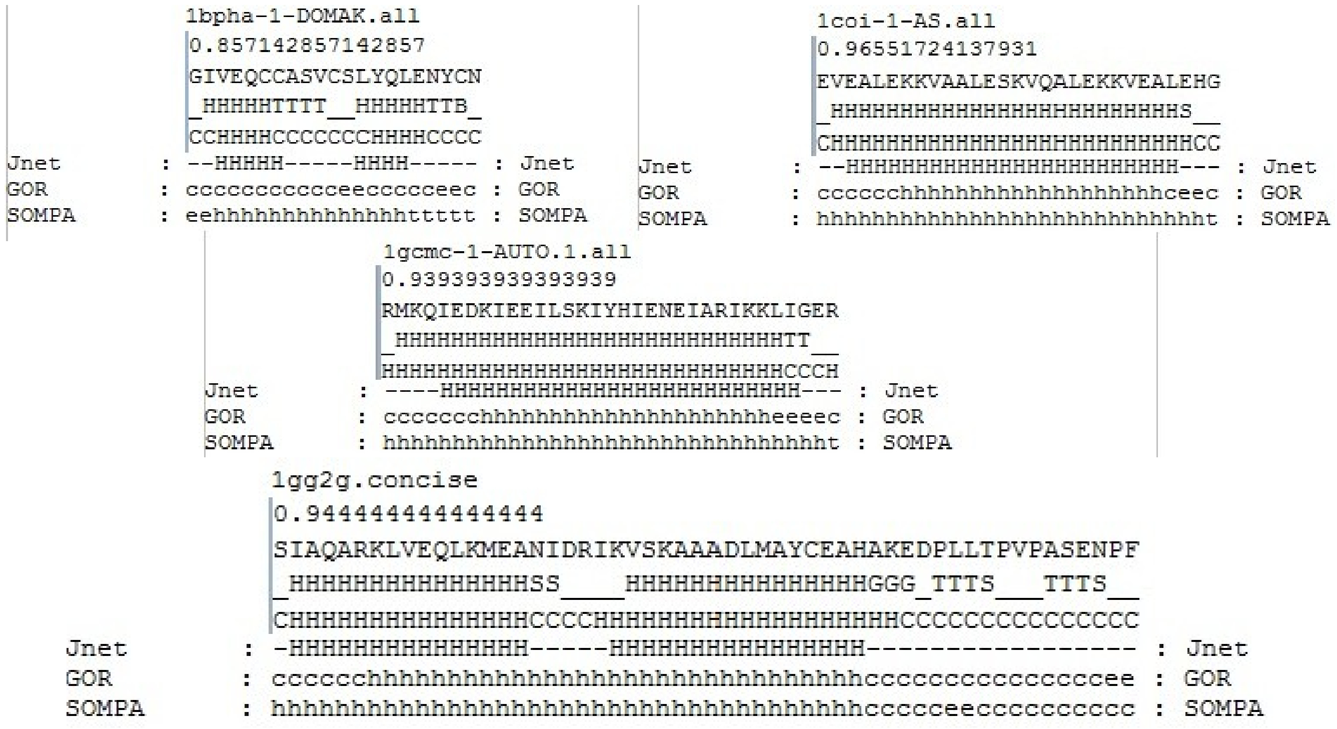


```
<terminated> 2ndstruc_motiftest.pl [Perl Local] Perl Interpreter
Choose Test:
0) HBB
1) CB513(training set)
2) CB406(testing set)

Choice: 2
43121/82069 (0.525423728813559)
```

Figure 17. Three Runs of 2ndstruc_motiftest. The above figures shows the runs of 2ndstruc_ motiftest which uses amino acids probabilities as well as neighboring amino acids to predict a secondary structure. The runs are on HBB, the CB513 training set, and the CB406 testing set.

Figure 17 shows runs of 2ndstruc_motiftest which takes probabilities of the amino acid and neighbors to predict what most likely structure the amino acid would take. Currently, the implementation is baised towards α-helices greatly over β-sheets. The method scores higher than the 2ndstruc_2aa method against the training and testing sets, but more importantly scores a lot higher on HBB than the other implemented methods. This method gives better results than GOR and SOMPA on the HBB protein.

The method can be further optimized. Since it currently focuses on α-helices, The method's poor performance comes from underestimating β-sheets. Highly α-helical proteins, such as 1coi in the CB513 set, scored as high as 96.5%, while proteins that were mostly β-sheets would score as low as 13.1%. The optimization could come in the form of having separate targets for β-sheets and coils. Notably, the method gives better results than the current implementation of Jpred3 on four proteins, three of which are from the CB513 training set and one is from the CB406 testing set. All three are short proteins with strong α-helical presence. The predicted results of the proteins are shown in the following figure as well as the predictions from Jpred, GOR, and SOPMA.



```
1bpha-1-DOMAK.all
0.857142857142857
GIVEQCCASVCSLYQLENYCN
_HHHHHTTTT__HHHHHTTB_
CCHHHHCCCCCCCHHHHCCCC
Jnet       : --HHHHH-----HHHH----- : Jnet
GOR        : ccccccccccceecccccceec : GOR
SOMPA      : eehhhhhhhhhhhhhhttttt : SOMPA
```

```
1coi-1-AS.all
0.96551724137931
EVEALEKKVAALESKVQALEKKVEALEHG
_HHHHHHHHHHHHHHHHHHHHHHHHHS__
CHHHHHHHHHHHHHHHHHHHHHHHHHHCC
Jnet       : --HHHHHHHHHHHHHHHHHHHHHHHH--- : Jnet
GOR        : cccccchhhhhhhhhhhhhhhhhhhceec : GOR
SOMPA      : hhhhhhhhhhhhhhhhhhhhhhhhhhhht : SOMPA
```

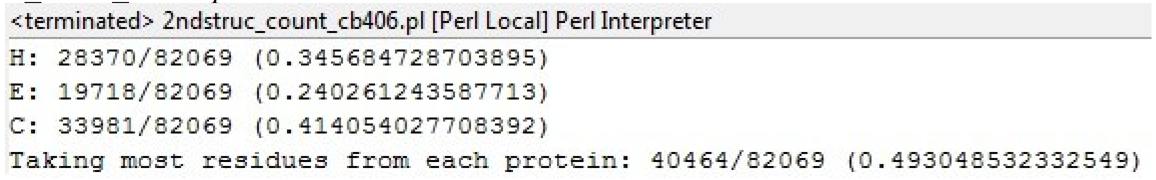
```
1gcmc-1-AUTO.1.all
0.939393939393939
RMKQIEDKIEEILSKIYHIENEIARIKKLIGER
_HHHHHHHHHHHHHHHHHHHHHHHHHHHHTT__
HHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCCH
Jnet       : ----HHHHHHHHHHHHHHHHHHHHHHHHHH--- : Jnet
GOR        : ccccccchhhhhhhhhhhhhhhhhhhhheeeec : GOR
SOMPA      : hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhht : SOMPA
```

```
1gg2g.concise
0.944444444444444
SIAQARKLVEQLKMEANIDRIKVSKAAADLMAYCEAHAKEDPLLTPVPASENPF
_HHHHHHHHHHHHHHHSS____HHHHHHHHHHHHHHHGGG_TTTS___TTTS__
CHHHHHHHHHHHHHHHHCCCHHHHHHHHHHHHHHHHHHHCCCCCCCCCCCCCCC
Jnet       : -HHHHHHHHHHHHHHHH-----HHHHHHHHHHHHHHHH---------------- : Jnet
GOR        : cccccchhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhccccccccccccccee : GOR
SOMPA      : hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhcccccceecccccccccc : SOMPA
```

Figure 18. Four Notable Results Displayed in Table. The figure displays the four proteins that are displayed in the table and the secondary structure prediction of the nearest neighbor implementation, Jpred, GOR, and SOPMA methods. The top three files are from the CB513 training set and the bottom file is from the CB406 testing set. Jnet was trained and tested on those sets.

Figure 18 displays four notable results that are produced by the nearest neighbor method. The four results each list the file, the accuracy of the implemented method, the primary structure, the secondary structure, the implementation's prediction, Jnet's prediction, GOR's prediction, and SOMPA's prediction. The secondary structure that is reported in each file uses the DSSP 8-class secondary structure system which can be converted to the 3-class by assuming H, G, and I are α-helices; B and E represent β-sheets; and all the other symbols represent random coils. The first object that stands out in all four runs is that they contain many α-helices which is in part the reason for which the implemented method performs well on these proteins. The current implementation favors α-helices regions over other regions. This allows for highly probable regions of secondary structure to be shown as well as force amino acids, to be part of bigger secondary structures if the amino acid is not part of a reasonable sized secondary structure. A final program was written to do some analysis on the CB406 testing set which is explained next.

2ndstruc_count_cb406.pl



```
<terminated> 2ndstruc_count_cb406.pl [Perl Local] Perl Interpreter
H: 28370/82069 (0.345684728703895)
E: 19718/82069 (0.240261243587713)
C: 33981/82069 (0.414054027708392)
Taking most residues from each protein: 40464/82069 (0.493048532332549)
```

Figure 19. Output of 2ndstruc_count_cb406. The above figure shows the output of 2ndstruc_count_cb406.pl which counts the number of helices, sheets, and coils that appear in the testing set.

Figure 19 was written to see how amino acid disruption to secondary structure is located in the testing set. It shows that about 35% of amino acids are in α-helices, 24% are in β-sheets, and 41% are in random coils. It is worth questioning if the distribution of amino acids to secondary structure represented by CB406 testing set relates to the natural distribution of amino acids or, more particularly, if different families of proteins have different distributions. The training sets and testing sets that are used on methods could change the results of the methods greatly. Currently, for *ab initio* approaches it is best to test for these proteins separately, but as more is understood about biology then it could be the case that every different type of protein included in a training set could be the best option. In the next chapters, we conclude with what the results mean and the future of *ab initio* protein structure prediction.

VI. Conclusion

Protein structure prediction is not an easy topic and the ability to accurately predict protein structures from a diverse set of amino acid sequences is still many years away. The importance of protein structure prediction and protein functions can not be denied. Even now, more and more protein sequences are being added to databases, while only a few are having their secondary structures and tertiary structures determined; however, as the sequence database expands, more information is available to refine and optimize computer structure prediction tools. Currently the best protein structure prediction methods are ones that find homologous sequences in databases and make alignments based on those. Unfortunately, often close to 50% of proteins lack viable homologues, which makes prediction methods based on homology impossible for some genomes.

Several methods were attempted and implemented to determine information that should be used and whether the primary structure of amino acids may influence the conformation of secondary structure of proteins. The methods attempted were separated into two different sessions: transitional methods involving probabilities of amino acids starting or stopping secondary structure regions and positional methods involving amino acids probabilities of being a specific secondary structure. The methods were trained and tested on the same data that Jpred used. Positional methods performed noticeably better than transitional methods in determining of secondary structure of proteins. The transitional methods used included more data, but require more work to function properly. In this work, it may be the reason that transitional methods performed so poorly was that the log odds were determined by dividing by 1/3, while a more accurate distribution is displayed at the top of this page. There are still many possibilities to try in regards to transitional methods that could lead to more knowledge in the way the biology of amino acids performs. Although transitional methods were not too successful in this work, one should not conclude that they are inadequate for protein structure prediction.

The positional methods implemented performed better than the transitional ones. The divergent of nearest neighbor, had the most success beating two tools, GOR and SOMPA as well as performing better on some proteins than Jpred. With more optimization, methods that involve working with amino acids sub-regions in predicting amino acids secondary structure could produce exceptional results as an *ab initio* method. Homology and threading may be the strongest tools to use for protein structure prediction, but without reliable *ab initio* protein structure prediction methods, there will always be proteins whose structures are not accurately predicted. Attempting more *ab initio* protein structure prediction methods may result in finding the information that is needed in understanding protein structure prediction. The information that is found could be the breakthrough that leads to 100% accuracy for protein structure prediction or it could be one of the many the 0.01% step increase in accuracy and with that there is still future work to be done.

VII. Future Work

There are many things that could be beneficial in the field of protein structure prediction. Some future work could include a tool for being able to build Cn3d files or making Cn3d file structure available. Having Cn3d file information ready to the public would not hugely effect the prediction of protein structure directly, but it would allow a global file type to be used as an input or output which would allow better coordination of efforts on tools. This would give would be the ability to view three-dimensional structures of proteins and make structure information to the broad range of scientists working to push the field forward.

Although advancements in the laboratory and biological approaches to protein structure are vital to the success of the field, it is not the only road that leads to more understanding in proteomics. A better understanding of amino acids and protein sequences can be discovered by trying different methods and parameters to discover and analyze what affects protein structure and why. Improvements in homology and threading modeling can be analyzed through data sets. A step that could be taken was mention before. A computer program could predict what structures a particular protein sequence is not likely to assume and eliminate those possibilities to approach a realistic prediction of structure, rather than trying to correctly predict a protein's structure from the start. Since there are 2 out of 3 possible secondary structures that an amino acid does not participate in, it is more probable to guess an incorrect structure than a correct one. Being able to determine what secondary structure an amino acid would not take could remove the available choices for what secondary structure to predict from three to two and improve accuracy with conjunction with other methods.

Another possible path in protein structure prediction is in the consideration that secondary structure and tertiary structure are more closely tied together. A possible route to take is to include how a tertiary structure relates to a secondary structure. Then there could be steps taken to make an initial guess of a secondary structure and then from the secondary structure predict a tertiary structure. The approach would then take the tertiary structure and make alterations to the secondary structure in an attempt to make it more stable. The future for computer scientists in protein structure prediction is not waiting for advancements in biology, but instead to exhaust every method to find every piece of information to advance biology. Attempting methods, like the previously mentioned, are important in learning more information about proteins outside of a lab.

Eventually, the goal is that a protein's final structure would be known from its primary sequence, its amino acid sequence, and then that would reveal the protein's function. Being able to obtain this kind of information would result in tremendous advances in medicine and scientific research.

VIII. Bibliography

Clair C. S., and Visick J. (2010). RNA and Protein Structure: Structure Prediction. *Exploring Bioinformatics: A Project-Based Approach, 1*(7), 197-230.

Alberts, Bray, Hopkin, Johnson, Lewis, Raff, Roberts, and Walter. (2004). Protein Structure and Function. *Essential Cell Biology, 2*(4), 119-157.

Chou P. Y., and Fasman G. D. (1974). Conformational Parameters for Amino Acids in Helical, β-Sheet, and Random Coil Regions Calculated from Proteins. *Biochemistry, 13*(2), 211-222.

Jones D. T. (1999). Protein Secondary Structure Prediction Based on Position-specific Scoring Matrices. *Journal of Molecular Biology, 292*(2), 195-202. doi:10.1006/jmbi.1999.3091

Cuff J. A., Clamp M. E., Siddiqui A. S., Finlay M. and Barton G. J. (1998). Jpred: A Consensus Secondary Structure Prediction Server. *Bioinformatics, 14*(10), 892-893.

Cuff, J. A., and Barton, G. J. (2000). Application of Enhanced Multiple Sequence Alignment Profiles to Improve Protein Secondary Structure Prediction. *PROTEINS: Structure, Function and Genetics 40,* 502-511.

Cole C., Barber J. D., and Barton G. J. (2008). The Jpred 3 secondary structure prediction server. *Nucleic Acids Research, 36*, W197–W201. doi:10.1093/nar/gkn238

Eddy S. R. (1998). Profile hidden Markov models. *Bioinformatics, 14*(9), 755-763.

Forney G. D. Jr. (1973). The Viterbi Algorithm. *Proceedings of the IEEE, 61*(3), 268-278. doi:10.1109/PROC.1973.9030

Altschul S. F., Madden T. L., Schäffer A. A., Zhang J., Zhang Z., Miller W., and Lipman D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research, 25*(17)*,* 3389-3402.

Suzek B. E., Huang H., McGarvey P., Mazumder R., and Wu C. H. (2007). UniRef: comprehensive and non-redundant UniProt reference clusters. *Bioinformatics, 23*(10), 1282–1288. doi:10.1093/bioinformatics/btm098

Simons K. T., Bonneau R., Ruczinski I., and Baker D. (1999). Ab Initio Protein Structure Prediction of CASP III Targets Using ROSETTA. *PROTEINS: Structure, Function, and Genetics, 37*(3), 171-176. doi: 10.1002/(SICI)1097-0134(1999)37:3+<171::AID-PROT21>3.0.CO;2-Z

Swiss Institute of Bioinformatics. (2011). *ExPASy Proteomics Server.* Retrieved from http://www.expasy.ch/tools/

BioGem.Org. (2009). *CFSSP: Chou & Fasman Secondary Structure Prediction Server.* Retrieved from http://www.biogem.org/tool/chou-fasman/

The Barton Group, - The University of Dundee. (2008). *Jpred – A Secondary Structure Prediction Server.* Retrieved from http://www.compbio.dundee.ac.uk/www-jpred/

Chen K., Kurgan L., and Ruan J. (2006). *Optimization of the Sliding Window Size for Protein Structure Prediction. Computational Intelligence and Bioinformatics and Computational Biology.* 28, 1–7. doi:10.1109/CIBCB.2006.330959

Bioinformatics Centre Institute of Microbial Technology. (2000). *APSSP: Advanced Protein Secondary Structure Prediction Server.* Retrieved from http://imtech.res.in/raghava/apssp/

Pole Bioinformatics Lyonnais. (2010). *GOR IV SECONDARY STRUCTURE PREDICTION METHOD.* Retrieved from http://npsa-pbil.ibcp.fr/cgi-bin/npsa_automat.pl?page=npsa_gor4.html

Pole Bioinformatics Lyonnais. (2010). *HNN.* Retrieved from http://npsa-pbil.ibcp.fr/cgi-bin/npsa_automat.pl?page=npsa_nn.html

Meiler Lab. (2011). *Jufo.* Retrieved from http://www.meilerlab.org/web/view.php?section=0&page=6

University College Dublin. (n.d.). *PORTER: Protein Secondary Structure Prediction at University College Dublin.* Retrieved from http://distill.ucd.ie/porter/

Aberystwyth University. (2003). *PROF - Secondary Structure Prediction System.* Retrieved from http://www.aber.ac.uk/~phiwww/prof/

University College London. (2008). *The PSIPRED Protein Structure Prediction Server.* Retrieved from http://bioinf.cs.ucl.ac.uk/psipred/

Pole Bioinformatics Lyonnais. (2010). *SOPMA SECONDARY STRUCTURE PREDICTION METHOD.* Retrieved from http://npsa-pbil.ibcp.fr/cgi-bin/npsa_automat.pl?page=npsa_sopma.html

University of California, Irvine. (n.d.). *SCRATCH Protein Predictor.* Retrieved from http://scratch.proteomics.ics.uci.edu/index.html

UW Baker Lab. (2011). *foldit: Solve Puzzles for Science.* Retrieved from http://fold.it/portal/

Madison Area Technical College. (n.d.). *The Biotechnology Project: Proteins.* ***Figure 1.*** Retrieved from http://biotech.matcmadison.edu/resources/proteins/labManual/chapter_2.htm

Imade Asemota Foundation. (2009). *Sickle Cell Disease.* **Figure 2.** Retrieved from http://www.imadefoundation.org/sickle-cell/

IX. Appendix

i. Instructions to Run Programs on a Windows Computer

1. One has to make sure that *Perl* is installed on the system. ActivePerl is one alternative to use.

2. Depending the on Perl install, Environmental Variables may need to be set. Listed in bullets points are reasons that they may not need to be set. If they do not need to be set, we have to skip to step 5.
   - The Perl installer specified that Environmental Variables or PATH variable has been set.
   - The Perl install has a GUI element which is used to run files.
   - The Perl install is in the same directory as where the Perl programs to run are.

3. If Environmental Variables need to be set, right click on My Computer and select properties, select advanced, select environment variables.

4. Select the PATH variable and click edit. Add a semicolon to the end if there isn't one there already and add the directory where the Perl interpreter is located for example *C:\Perl\bin*.

5. Navigate to the folder through the command line in which the program to be run is located.

6. Make sure all required input files for the program are present unless otherwise specified.

7. Run the program by typing the perl interpreter followed by the name of the file. An example would be, *perl 2ndstruc_choufasman.pl* to run the 2ndstruc_choufasman program.

ii. Code

```
my @filelist, @m, @m2, @tr_m, @i_m, @tr_20m;
my %aa, %struc;
$aa{"A"} = 0; $aa{"a"} = 0;
$aa{"C"} = 1; $aa{"c"} = 1;
$aa{"D"} = 2; $aa{"d"} = 2;
$aa{"E"} = 3; $aa{"e"} = 3;
$aa{"F"} = 4; $aa{"f"} = 4;
$aa{"G"} = 5; $aa{"g"} = 5;
$aa{"H"} = 6; $aa{"h"} = 6;
$aa{"I"} = 7; $aa{"i"} = 7;
$aa{"K"} = 8; $aa{"k"} = 8;
$aa{"L"} = 9; $aa{"l"} = 9;
$aa{"M"} = 10;$aa{"m"} = 10;
$aa{"N"} = 11;$aa{"n"} = 11;
$aa{"P"} = 12;$aa{"p"} = 12;
$aa{"Q"} = 13;$aa{"q"} = 13;
$aa{"R"} = 14;$aa{"r"} = 14;
$aa{"S"} = 15;$aa{"s"} = 15;
$aa{"T"} = 16;$aa{"t"} = 16;
$aa{"V"} = 17;$aa{"v"} = 17;
$aa{"W"} = 18;$aa{"w"} = 18;
$aa{"Y"} = 19;$aa{"y"} = 19;

#helix (G, H and I), strand (E and B) and loop (all others).
$struc{"G"} = 0;$struc{"H"} = 0;$struc{"I"} = 0;
$struc{"E"} = 1;$struc{"B"} = 1;

for(my $i=0; $i<20; $i++){
    for(my $j=0; $j<3; $j++){
        $m[$i][$j] = 0;
        $m2[$i][$j] = 0;
    }
}

for(my $i=0; $i<3; $i++){
    $i_m[$i] = 0;
    for(my $j=0; $j<3; $j++){
        $tr_m[$i][$j] = 0;
    }
}

for(my $i=0; $i<60; $i++){
    for(my $j=0; $j<60; $j++){
        $tr_20m[$i][$j] = 1;
    }
}

#printf "Enter the filename: ";
#my $file = <STDIN>;
#chomp $file;
$dir = "513_distribute/";

open(INFILE, "513_distribute/list.txt") || die("Could not open file!");
@filelist = <INFILE>;
close(INFILE);
#print $filelist[0] . "\n";

#$file = "1atpi-1-DOMAK.all";
#$file = "1541-1-AUTO.1.all";
```

```
foreach my $file(@filelist){
    open(INFILE, $dir.$file) || die("Could not open file!");
    @info = <INFILE>;
    close(INFILE);

    chomp $info[0];
    chomp $info[1];
    chomp $info[5];

    @line1 = split(/,/, substr($info[0], 4));
    @line2 = split(/,/, substr($info[1], 5));
    @line3 = split(/,/, substr($info[5], 7));

    my $i = 0;
    my $prev = 0;
    my $prev_letter;

    if(exists $struc{$line2[0]}){
            $i_m[$struc{$line2[0]}] = $i_m[$struc{$line2[0]}] + 1;
    }
    else{
        $i_m[2] = $i_m[2] + 1;
    }
    foreach my $letter(@line1){
        $temp = 2;
        if(exists $struc{$line3[$i]}){
            $temp = $struc{$line3[$i]};
        }
        $m2[$aa{$letter}][$temp] = $m2[$aa{$letter}][$temp] + 1;

        $temp = 2;
        if(exists $struc{$line2[$i]}){
            $temp = $struc{$line2[$i]};
        }
        $m[$aa{$letter}][$temp] = $m[$aa{$letter}][$temp] + 1;
        if($i > 0){
            $tr_m[$prev][$temp] = $tr_m[$prev][$temp] + 1;
            $tr_20m[$aa{$prev_letter}*3+$prev][$aa{$letter}*3+$temp]
 =$tr_20m[$aa{$prev_letter}*3+$prev][$aa{$letter}*3+$temp] + 1;
        }
        $i++;
        $prev = $temp;
        $prev_letter = $letter;
    }
}

#print "Initation:\n";
open(OUTFILE, ">1imatrix.txt") || die("Could not open file!");
for(my $i=0; $i<3; $i++){
    print OUTFILE $i_m[$i] . " ";
}
#print "\n\n";
close(OUTFILE);

#print "Transition:\n";
open(OUTFILE, ">1tmatrix.txt") || die("Could not open file!");
for(my $i=0; $i<3; $i++){
    for(my $j=0; $j<3; $j++){
        print OUTFILE $tr_m[$i][$j] . " ";
    print OUTFILE "\n";
}
#print "\n\n";
close(OUTFILE);

#print "Transition 20:\n";
open(OUTFILE, ">1t20matrix.txt") || die("Could not open file!");
for(my $i=0; $i<60; $i++){
    for(my $j=0; $j<60; $j++){
        print OUTFILE $tr_20m[$i][$j] . " ";
    }
    print OUTFILE "\n";
}
#print "\n\n";
close(OUTFILE);

#print "Emission:\n";
open(OUTFILE, ">1matrix.txt") || die("Could not open file!");
open(OUTFILE2, ">1matrix2.txt") || die("Could not open file!");
for(my $i=0; $i<20; $i++){
    for(my $j=0; $j<3; $j++){
        print OUTFILE $m[$i][$j] . " ";
        print OUTFILE2 $m2[$i][$j] . " ";
    }
    print OUTFILE "\n";
    print OUTFILE2 "\n";
}
close(OUTFILE);
close(OUTFILE2);

open(OUTFILE, ">1pmatrix.txt") || die("Could not open file!");
for(my $i=0; $i<20; $i++){
    my $temp = 0;
    for(my $j=0; $j<3; $j++){
        $temp = $temp + $m[$i][$j];
    }
    for(my $j=0; $j<3; $j++){
        print OUTFILE sprintf("%.3f", $m[$i][$j]/$temp) . " ";
    }
    print OUTFILE "\n";
}
close(OUTFILE);


exit;
```

```
my @file, @matrix, @log_matrix, @tmatrix, @log_tmatrix, @imatrix, @log_imatrix;
my %aa, @struc, %input;
$aa{"A"} = 0; $aa{"a"} = 0;
$aa{"C"} = 1; $aa{"c"} = 1;
$aa{"D"} = 2; $aa{"d"} = 2;
$aa{"E"} = 3; $aa{"e"} = 3;
$aa{"F"} = 4; $aa{"f"} = 4;
$aa{"G"} = 5; $aa{"g"} = 5;
$aa{"H"} = 6; $aa{"h"} = 6;
$aa{"I"} = 7; $aa{"i"} = 7;
$aa{"K"} = 8; $aa{"k"} = 8;
$aa{"L"} = 9; $aa{"l"} = 9;
$aa{"M"} = 10;$aa{"m"} = 10;
$aa{"N"} = 11;$aa{"n"} = 11;
$aa{"P"} = 12;$aa{"p"} = 12;
$aa{"Q"} = 13;$aa{"q"} = 13;
$aa{"R"} = 14;$aa{"r"} = 14;
$aa{"S"} = 15;$aa{"s"} = 15;
$aa{"T"} = 16;$aa{"t"} = 16;
$aa{"V"} = 17;$aa{"v"} = 17;
$aa{"W"} = 18;$aa{"w"} = 18;
$aa{"Y"} = 19;$aa{"y"} = 19;
$struc[0] = 'H'; $struc[1] = 'E'; $struc[2] = 'C';
$input{"G"} = 0; $input{"H"} = 0; $input{"I"} = 0;
$input{"E"} = 1; $input{"B"} = 1;

if($file eq ""){
    $file="1.cgi";
}

my $protein =
 "VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFAT
LSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH";
my $result =
 "CCCCHHHHHHHHHHHCCCCHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHCCCCCCCCCCCHH
HHHHHHCCCCCCCHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHHHHHHHHHHCCCC";

if(-e "p1matrix.txt"){
    open(INFILE, "p1matrix.txt") || die("Could not open file!");
    @file = <INFILE>;
    close(INFILE);

    for(my $i=0; $i<@file; $i++){
        chomp $file[$i];
        my @temp = split(/ /, $file[$i]);
        for(my $j=0; $j<@temp; $j++){
            $log_matrix[$i][$j] = $temp[$j];
        }
    }
}
else {
    open(INFILE, "1matrix.txt") || die("1matrix.txt not found!");
    @file = <INFILE>;
    close(INFILE);

    for(my $i=0; $i<@file; $i++){
    chomp $file[$i];
        my @temp = split(/ /, $file[$i]);
        for(my $j=0; $j<@temp; $j++){
```

```
        foreach my $val(@$row){
            $temp += $val;
        }

        foreach my $val(@$row){
            $val = log(($val/$temp)/(1/3));
        }
    }

    open(OUTFILE, ">p1matrix.txt") || die("Could not open file!");
    for(my $i=0; $i<20; $i++){
        for(my $j=0; $j<3; $j++){
            print OUTFILE $log_matrix[$i][$j] . " ";
        }
        print OUTFILE "\n";
    }
    close(OUTFILE);
}


print "Choose Test: \n";
print "0) HBB\n";
print "1) CB513(training set)\n";
print "2) CB406(testing set)\n\n";
print "Choice: ";
my $test = <STDIN>;

if($test == 0){
    my $choice, $chance;
my $sstruc = '';



for(my $i=0; $i<length($protein); $i++){
    $choice = 0;
    $chance = $log_matrix[$aa{substr($protein, $i, 1)}][0];
    for(my $j=1; $j<3; $j++){
        my $temp = $log_matrix[$aa{substr($protein, $i, 1)}][$j];
        if($temp > $chance){
            $choice = $j;
            $chance = $temp;

        }
    }
    $sstruc = $sstruc . $choice;
}

$temp = "";
for(my $i=0; $i<length($sstruc); $i++){
    $temp =  $temp . $struc[substr($sstruc, $i, 1)];
}

print $sstruc . "\n";
print $temp . "\n";
print $result . "\n";

my $score = 0;
for(my $i=0; $i<length($result); $i++){
    if(substr($temp, $i, 1) eq substr($result, $i, 1)){
        $score++;
    }
}

print $score . "/" . length($result) . " (" . $score/length($result) . ")\n";
```

```
}
elsif($test==1){
    open(INFILE, "513_distribute/list.txt") || die("Could not open file!");
@filelist = <INFILE>;
close(INFILE);

my $dir = "513_distribute/";

    my $score1 = 0;
    my $score2 = 0;
    my $total = 0;

foreach my $file(@filelist){
    open(INFILE, $dir.$file) || die("Could not open file!");
    @info = <INFILE>;
    close(INFILE);

    chomp $info[0];
    chomp $info[1];
    chomp $info[5];

    @line1 = split(/,/, substr($info[0], 4));
    @line2 = split(/,/, substr($info[1], 5));
    @line3 = split(/,/, substr($info[5], 7));

    my $correct1 = 0;
    my $correct2 = 0;
    my $choice, $chance;
    my $sstruc = "";


    my $temp = 0;
    my $i = 0;
    foreach my $letter(@line1){
        $choice = 0;
        $chance = $log_matrix[$aa{$letter}][0];
        for(my $j=1; $j<3; $j++){
            my $temp = $log_matrix[$aa{$letter}][$j];
            if($temp > $chance){
                $choice = $j;
                $chance = $temp;
            }
        }
        $sstruc = $sstruc . $choice;
    }

    for($i=0; $i<length($sstruc); $i++){
        my $check = 2;
        if(exists $input{$line2[$i]}){
            $check = $input{$line2[$i]};
        }
        if(substr($sstruc, $i, 1) == $check){
            $correct1++;
        }
         $check = 2;
        if(exists $input{$line3[$i]}){
            $check = $input{$line3[$i]};
        }
        if(substr($sstruc, $i, 1) == $check){
            $correct2++;
        }
    }
    $score1 = $score1 + $correct1;
    $score2 = $score2 + $correct2;
```

```
        $total = $total + $i;
        #last;

    }
        print $score1 . "/" . $total . " (" . $score1/$total . ")\n";
        print $score2 . "/" . $total . " (" . $score2/$total . ")\n";

    }
    elsif($test == 2){
        open(INFILE, "406_distribute//list.txt") || die("Could not open file!");
        @filelist = <INFILE>;
        close(INFILE);

    my $dir = "406_distribute/";

        my $score1 = 0;
        my $score2 = 0;
        my $total = 0;

    foreach my $file(@filelist){
        open(INFILE, $dir.$file) || die("Could not open file!");
        @info = <INFILE>;
        close(INFILE);

        chomp $info[0];
        chomp $info[scalar @info - 2];

        @line1 = split(/,/, substr($info[0], 20));
        @line2 = split(/,/, substr($info[scalar @info - 2], 5));



        my $correct1 = 0;
        my $correct2 = 0;
        my $choice, $chance;
        my $sstruc = "";


        my $temp = 0;
        my $i = 0;
        foreach my $letter(@line1){
            $choice = 0;
            $chance = $log_matrix[$aa{$letter}][0];
            for(my $j=1; $j<3; $j++){
                my $temp = $log_matrix[$aa{$letter}][$j];
                if($temp > $chance){
                    $choice = $j;
                    $chance = $temp;
                }
            }
            $sstruc = $sstruc . $choice;
        }

        for($i=0; $i<length($sstruc); $i++){
            my $check = 2;
            if(exists $input{$line2[$i]}){
                $check = $input{$line2[$i]};
            }
            if(substr($sstruc, $i, 1) == $check){
                $correct1++;
            }
        }
        $score1 = $score1 + $correct1;
        $total = $total + $i;
    #last;
}
    print $score1 . "/" . $total . " (" . $score1/$total . ")\n";
}
exit;
```

```
my @file, @matrix, @log_matrix, @tmatrix, @log_tmatrix, @imatrix, @log_imatrix,
 @t20matrix, @log_t20matrix,;
my %aa, @struc, %input;
$aa{"A"} = 0; $aa{"a"} = 0;
$aa{"C"} = 1; $aa{"c"} = 1;
$aa{"D"} = 2; $aa{"d"} = 2;
$aa{"E"} = 3; $aa{"e"} = 3;
$aa{"F"} = 4; $aa{"f"} = 4;
$aa{"G"} = 5; $aa{"g"} = 5;
$aa{"H"} = 6; $aa{"h"} = 6;
$aa{"I"} = 7; $aa{"i"} = 7;
$aa{"K"} = 8; $aa{"k"} = 8;
$aa{"L"} = 9; $aa{"l"} = 9;
$aa{"M"} = 10;$aa{"m"} = 10;
$aa{"N"} = 11;$aa{"n"} = 11;
$aa{"P"} = 12;$aa{"p"} = 12;
$aa{"Q"} = 13;$aa{"q"} = 13;
$aa{"R"} = 14;$aa{"r"} = 14;
$aa{"S"} = 15;$aa{"s"} = 15;
$aa{"T"} = 16;$aa{"t"} = 16;
$aa{"V"} = 17;$aa{"v"} = 17;
$aa{"W"} = 18;$aa{"w"} = 18;
$aa{"Y"} = 19;$aa{"y"} = 19;
$struc[0] = 'H'; $struc[1] = 'E'; $struc[2] = 'C';
$input{"G"} = 0; $input{"H"} = 0; $input{"I"} = 0;
$input{"E"} = 1; $input{"B"} = 1;

if($file eq ""){
    $file="1.cgi";
}

$protein =
 "VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTF
ATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH";
$result =
 "CCCCHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHCCCCCCCCCCC
HHHHHHHHHCCCCCCCHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHHHHHHHHHCCCC";
if(-e "p1matrix.txt"){
    open(INFILE, "p1matrix.txt") || die("Could not open file!");
    @file = <INFILE>;
    close(INFILE);

    for(my $i=0; $i<@file; $i++){
        chomp $file[$i];
        my @temp = split(/ /, $file[$i]);
        for(my $j=0; $j<@temp; $j++){
            $log_matrix[$i][$j] = $temp[$j];
        }
    }
}
else {
    open(INFILE, "1matrix.txt") || die("1matrix.txt not found!");
    @file = <INFILE>;
    close(INFILE);

    for(my $i=0; $i<@file; $i++){
    chomp $file[$i];
        my @temp = split(/ /, $file[$i]);
        for(my $j=0; $j<@temp; $j++){
            $matrix[$i][$j] = $temp[$j];
            $log_matrix[$i][$j] = $temp[$j];
        }
    }
```

```
            $temp += $val;
        }

        foreach my $val(@$row){
            $val = log(($val/$temp)/(1/3));
        }
    }

    open(OUTFILE, ">p1matrix.txt") || die("Could not open file!");
    for(my $i=0; $i<20; $i++){
        for(my $j=0; $j<3; $j++){
            print OUTFILE $log_matrix[$i][$j] . " ";
        }
        print OUTFILE "\n";
    }
    close(OUTFILE);
}


if(-e "p1imatrix.txt"){
    open(INFILE, "p1imatrix.txt") || die("1imatrix.txt not found!");
    @file = <INFILE>;
    close(INFILE);

    my @temp = split(/ /, $file[0]);
    for(my $j=0; $j<@temp; $j++){
        $log_imatrix[$j] = $temp[$j];
    }
}
else{
    open(INFILE, "1imatrix.txt") || die("1imatrix.txt not found!");
    @file = <INFILE>;
    close(INFILE);

    my @temp = split(/ /, $file[0]);
    for(my $j=0; $j<@temp; $j++){
        $imatrix[$j] = $temp[$j];
        $log_imatrix[$j] = $temp[$j];
    }

    my $temp = 0;
    #converts log-odds variable into log-odds
    foreach my $val(@log_imatrix){
        $temp += $val;
    }
    foreach my $val(@log_imatrix){
        $val = log(($val/$temp)/(1/3));
    }

    open(OUTFILE, ">p1imatrix.txt") || die("Could not open file!");
    for(my $i=0; $i<3; $i++){
        print OUTFILE $log_imatrix[$i] . " ";
    }
    close(OUTFILE);
}

if(-e "p1tmatrix.txt"){
    open(INFILE, "p1tmatrix.txt") || die("1tmatrix.txt not found!");
    @file = <INFILE>;
    close(INFILE);

    for(my $i=0; $i<@file; $i++){
        chomp $file[$i];
        my @temp = split(/ /, $file[$i]);
        for(my $j=0; $j<@temp; $j++){
            $log_tmatrix[$i][$j] = $temp[$j];
        }
    }
}
else {
    open(INFILE, "1tmatrix.txt") || die("1tmatrix.txt not found!");
    @file = <INFILE>;
    close(INFILE);

    for(my $i=0; $i<@file; $i++){
        chomp $file[$i];
        my @temp = split(/ /, $file[$i]);
        for(my $j=0; $j<@temp; $j++){
            $tmatrix[$i][$j] = $temp[$j];
            $log_tmatrix[$i][$j] = $temp[$j];
        }
    }

    my $temp;
    #converts log-odds variable into log-odds
    foreach my $row(@log_tmatrix){
        $temp = 0;
        foreach my $val(@$row){
            $temp += $val;
        }
        foreach my $val(@$row){
            $val = log(($val/$temp)/(1/3));
        }
    }

    open(OUTFILE, ">p1tmatrix.txt") || die("Could not open file!");
    for(my $i=0; $i<3; $i++){
        for(my $j=0; $j<3; $j++){
            print OUTFILE $log_tmatrix[$i][$j] . " ";
        }
        print OUTFILE "\n";
    }
    close(OUTFILE);
}


open(INFILE, "1t20matrix.txt") || die("Could not open file!");
@file = <INFILE>;
close(INFILE);

for(my $i=0; $i<@file; $i++){
    chomp $file[$i];
    my @temp = split(/ /, $file[$i]);
    for(my $j=0; $j<@temp; $j++){
        $t20matrix[$i][$j] = $temp[$j];
        $log_t20matrix[$i][$j] = $temp[$j];
    }

}

my $temp;
#converts log-odds variable into log-odds


#foreach my $row(@log_t20matrix){
#   $temp = 0;
#   foreach my $val(@$row){
#       $temp += $val;
#    }

#    foreach my $val(@$row){
#        $val = log(($val/$temp)/(1/60));
#    }
#}


#for(my $i=0; $i<$#matrix; $i++){
#    for(my $j=0; $j<$#matrix; $j++){
#        print $matrix[$i][$j] . " ";
#    }
#    print "\n";
#}
#print "\n\n";




#open(OUTFILE, ">p1tmatrix.txt") || die("Could not open file!");
#for(my $i=0; $i<3; $i++){
#    for(my $j=0; $j<3; $j++){
#        print OUTFILE $log_tmatrix[$i][$j] . " ";
#    }
#    print OUTFILE "\n";
#}
##print "\n\n";
#close(OUTFILE);




#my $choice, $chance;
#my $sstruc = '';

#$choice = 0;
#$chance = $log_imatrix[0] + $log_matrix[$aa{substr($protein, 0, 1)}][0];
#for(my $j=1; $j<3; $j++){
#    my $temp = $log_imatrix[$j] + $log_matrix[$aa{substr($protein, 0, 1)}][$j];
#    if($temp > $chance){
#        $choice = $j;
#        $chance = $temp;
#    }
#}
#$sstruc = $sstruc . $choice;


#for(my $i=1; $i<length($protein); $i++){
#    $choice = 0;
#    $chance = $log_tmatrix[substr($sstruc, $i-1,1)][0] +
$log_matrix[$aa{substr($protein, $i, 1)}][0];
#    #$chance = $log_t20matrix[$aa{substr($protein, $i-1, 1)}*3+substr($sstruc, $i-1,1)]
[$aa{substr($protein, $i, 1)}*3] + $log_matrix[$aa{substr($protein, $i, 1)}][0];
#    for(my $j=1; $j<3; $j++){
#        my $temp = $log_tmatrix[substr($sstruc, $i-1,1)][$j] +
$log_matrix[$aa{substr($protein, $i, 1)}][$j];
#        #my $temp = $log_t20matrix[$aa{substr($protein, $i-1, 1)}*3+substr($sstruc,
$i-1,1)][$aa{substr($protein, $i, 1)}*3+$j] + $log_matrix[$aa{substr($protein, $i, 1)}]
[$j];
#        if($temp > $chance){
#            $choice = $j;
#            $chance = $temp;
#        }
#    }
```

```
print "1) CB513(training set)\n";
print "2) CB406(testing set)\n\n";
print "Choice: ";
my $test = <STDIN>;

if($test == 0){
    my $choice, @table, @path;
    my $choice;
    my $chance;

    for(my $j=0; $j<3; $j++){
        $path[0][$j] = "" . $j;
        $table[0][$j] = $log_imatrix[$j] + $log_matrix[$aa{substr($protein, 0,
 1)}][$j];
    }

    for(my $i=1; $i<length($protein); $i++){
        for(my $j=0; $j<3; $j++){
            $choice = 0;
            $chance = $table[$i-1][0] + $log_tmatrix[0][$j] +
 $log_matrix[$aa{substr($protein, $i, 1)}][0];
            #print $chance . " = " . $table[$i-1][0] . " + " . $log_tmatrix[0][$j] . "
+ " . $log_matrix[$aa{substr($protein, $i, 1)}][0] . "\n";
            for(my $k=1; $k<3; $k++){
                my $temp = $table[$i-1][$k] + $log_tmatrix[$k][$j] +
 $log_matrix[$aa{substr($protein, $i, 1)}][$k];
                #print $temp . " = " . $table[$i-1][$k] . " + " . $log_tmatrix[$k][$j]
. " + " . $log_matrix[$aa{substr($protein, $i, 1)}][$k] . "\n";
                if($temp > $chance){
                    $choice = $k;
                    $chance = $temp;
                }
            }
            $path[$i][$j] = $path[$i-1][$choice] . $j;
            $table[$i][$j] = $chance;
        }
    }


    $temp = "";
    print $protein . "\n";
    print $result . "\n";
    for(my $i=0; $i<3; $i++){
        print $path[length($protein)-1][$i] . "\n";
        #print $table[length($protein)-1][$i] . "\n";
    }
}
elsif($test == 1){
    open(INFILE, "513_distribute/list.txt") || die("Could not open file!");
@filelist = <INFILE>;
close(INFILE);

my $dir = "513_distribute/";

    my $score1 = 0;
    my $score2 = 0;
    my $total = 0;

foreach my $file(@filelist){
    open(INFILE, $dir.$file) || die("Could not open file!");
    @info = <INFILE>;
    close(INFILE);

    chomp $info[0];
    @line3 = split(/,/, substr($info[5], 7));



    my $correct1 = 0;
    my $correct2 = 0;
    my $choice, $chance, $struc_chance = 0;
    my $sstruc = "";
    my $sstruc2 = "";
    my $sstruc3 = "";


    my $temp = 0;
    my $i = 0;
    foreach my $letter(@line1){
        for(my $j=0; $j<3; $j++){
            $path[0][$j] = "" . $j;
            $table[0][$j] = $log_imatrix[$j] + $log_matrix[$aa{$letter}][$j];
        }
            for(my $j=0; $j<3; $j++){
                $choice = 0;
                $chance = $table[$i-1][0] + $log_tmatrix[0][$j] +
$log_matrix[$aa{$letter}][0];
                for(my $k=1; $k<3; $k++){
                    my $temp = $table[$i-1][$k] + $log_tmatrix[$k][$j] +
$log_matrix[$aa{$letter}][$k];
                    if($temp > $chance){
                        $choice = $k;
                        $chance = $temp;
                    }
                }

            $path[$i][$j] = $path[$i-1][$choice] . $j;
            $table[$i][$j] = $chance;
            }





        $i++;
        $sstruc = $sstruc . $choice;
    }

    $choice = 0;
    if($table[$i-1][1] > $table[$i-1][$choice]){
        $choice = 1;
    }
    if($table[$i-1][2] > $table[$i-1][$choice]){
        $choice = 2;
    }

    for(my $j=0; $j<$i; $j++){
        my $check = 2;
        if(exists $input{$line2[$j]}){
            $check = $input{$line2[$j]};
        }
        if(substr($path[$i-1][$choice], $j,1) == $check){
            $correct1++;
        }

        my $check = 2;
        if(exists $input{$line3[$j]}){
            $check = $input{$line3[$j]};
```

```
                $correct2++;
            }
        }
        $score1 = $score1 + $correct1;
        $score2 = $score2 + $correct2;
        $total = $total + $i;
        #last;

    }
        print $score1 . "/" . $total . " (" . $score1/$total . ")\n";
        print $score2 . "/" . $total . " (" . $score2/$total . ")\n";
    }
    elsif($test == 2){
        open(INFILE, "406_distribute//list.txt") || die("Could not open file!");
        @filelist = <INFILE>;
        close(INFILE);

    my $dir = "406_distribute/";

        my $score1 = 0;
        my $score2 = 0;
        my $total = 0;

    foreach my $file(@filelist){
        open(INFILE, $dir.$file) || die("Could not open file!");
        @info = <INFILE>;
        close(INFILE);

        chomp $info[0];
        chomp $info[scalar @info - 2];

        @line1 = split(/,/, substr($info[0], 20));
        @line2 = split(/,/, substr($info[scalar @info - 2], 5));



        my $correct1 = 0;
        my $correct2 = 0;
        my $choice, $chance, $struc_chance = 0;
        my $sstruc = "";
        my $sstruc2 = "";
        my $sstruc3 = "";


        my $temp = 0;
        my $i = 0;
        foreach my $letter(@line1){
            for(my $j=0; $j<3; $j++){
                $path[0][$j] = "" . $j;
                $table[0][$j] = $log_imatrix[$j] + $log_matrix[$aa{$letter}][$j];
            }
                for(my $j=0; $j<3; $j++){
                    $choice = 0;
                    $chance = $table[$i-1][0] + $log_tmatrix[0][$j] +
 $log_matrix[$aa{$letter}][0];
                    for(my $k=1; $k<3; $k++){
                        my $temp = $table[$i-1][$k] + $log_tmatrix[$k][$j] +
 $log_matrix[$aa{$letter}][$k];
                        if($temp > $chance){
                            $choice = $k;
                            $chance = $temp;
                    }
                }
            }
        $i++;
        $sstruc = $sstruc . $choice;
    }
    $choice = 0;
    if($table[$i-1][1] > $table[$i-1][$choice]){
        $choice = 1;
    }
    if($table[$i-1][2] > $table[$i-1][$choice]){
        $choice = 2;
    }

    for(my $j=0; $j<$i; $j++){
        my $check = 2;
        if(exists $input{$line2[$j]}){
            $check = $input{$line2[$j]};

        }
        #print $check;
        if(substr($path[$i-1][$choice], $j,1) == $check){
            $correct1++;
        }

    }
    #print "\n";
    #print $path[$i-1][$choice] . "\n";
    $score1 = $score1 + $correct1;
    $total = $total + $i;
    #last;

}
    print $score1 . "/" . $total . " (" . $score1/$total . ")\n";
}

exit;
```

```
my @file, @matrix, @log_matrix;
my %aa, @struc, %input;
$aa{"A"} = 0; $aa{"a"} = 0;
$aa{"C"} = 1; $aa{"c"} = 1;
$aa{"D"} = 2; $aa{"d"} = 2;
$aa{"E"} = 3; $aa{"e"} = 3;
$aa{"F"} = 4; $aa{"f"} = 4;
$aa{"G"} = 5; $aa{"g"} = 5;
$aa{"H"} = 6; $aa{"h"} = 6;
$aa{"I"} = 7; $aa{"i"} = 7;
$aa{"K"} = 8; $aa{"k"} = 8;
$aa{"L"} = 9; $aa{"l"} = 9;
$aa{"M"} = 10;$aa{"m"} = 10;
$aa{"N"} = 11;$aa{"n"} = 11;
$aa{"P"} = 12;$aa{"p"} = 12;
$aa{"Q"} = 13;$aa{"q"} = 13;
$aa{"R"} = 14;$aa{"r"} = 14;
$aa{"S"} = 15;$aa{"s"} = 15;
$aa{"T"} = 16;$aa{"t"} = 16;
$aa{"V"} = 17;$aa{"v"} = 17;
$aa{"W"} = 18;$aa{"w"} = 18;
$aa{"Y"} = 19;$aa{"y"} = 19;
$struc[0] = 'HH'; $struc[1] = 'HE'; $struc[2] = 'HC';
$struc[3] = 'EH'; $struc[4] = 'EE'; $struc[5] = 'EC';
$struc[6] = 'CH'; $struc[7] = 'CE'; $struc[8] = 'CC';
$input{"G"} = 0; $input{"H"} = 0; $input{"I"} = 0;
$input{"E"} = 1; $input{"B"} = 1;

if($file eq ""){
    $file="1.cgi";
}

$protein =
 "VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTF
ATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH";
my $result =
 "CCCCHHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHCCCCCCCCCC
HHHHHHHHHCCCCCCCHHHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHHHHHHHHHHHCCCC";


if(-e "p2matrix.txt"){
    open(INFILE, "p2matrix.txt") || die("Could not open file!");
    @file = <INFILE>;
    close(INFILE);

    for(my $i=0; $i<@file; $i++){
        chomp $file[$i];
        my @temp = split(/ /, $file[$i]);
        for(my $j=0; $j<@temp; $j++){
            $log_matrix[$i][$j] = $temp[$j];
        }
    }
}
else {
    open(INFILE, "2matrix.txt") || die("Could not open file!");
    @file = <INFILE>;
    close(INFILE);

    for(my $i=0; $i<@file; $i++){
        chomp $file[$i];
        my @temp = split(/ /, $file[$i]);
        for(my $j=0; $j<@temp; $j++){
            $matrix[$i][$j] = $temp[$j] + 1;
            $log_matrix[$i][$j] = $temp[$j] + 1;
    my $temp;
    #converts log-odds variable into log-odds
    foreach my $row(@log_matrix){
        $temp = 0;
        foreach my $val(@$row){
            $temp += $val;
        }

        foreach my $val(@$row){
            $val = log(($val/$temp)/(1/9));
        }
    }


    open(OUTFILE, ">p2matrix.txt") || die("Could not open file!");
    for(my $i=0; $i<400; $i++){
        for(my $j=0; $j<9; $j++){
            print OUTFILE $log_matrix[$i][$j] . " ";
        }
        print OUTFILE "\n";
    }
    close(OUTFILE);
}




#for(my $i=0; $i<$#matrix; $i++){
#   for(my $j=0; $j<$#matrix; $j++){
#       print $matrix[$i][$j] . " ";
#   }
#   print "\n";
#}
#print "\n\n";

print "Choose Test: \n";
print "0) HBB\n";
print "1) CB513(training set)\n";
print "2) CB406(testing set)\n\n";
print "Choice: ";
my $test = <STDIN>;

if($test == 0){
my $choice, $chance, $struc_chance = 0;
my $sstruc = "";


for(my $i=0; $i<length($protein); $i=$i+2){
    $choice = 0;
    $chance = $log_matrix[$aa{substr($protein, $i, 1)}*20 + $aa{substr($protein, $i+1,
 1)}][0];
    for(my $j=1; $j<9; $j++){
        my $temp = $log_matrix[$aa{substr($protein, $i, 1)}*20 + $aa{substr($protein,
 $i+1, 1)}][$j];
        if($temp > $chance){
            $choice = $j;
            $chance = $temp;
        }
    }
    $sstruc = $sstruc . $choice;
    $struc_chance = $struc_chance + $chance;
}
```

```
for(my $i=0; $i<length($sstruc); $i++){
    $temp =  $temp . $struc[substr($sstruc, $i, 1)];
}

#print $sstruc . "\n";
print $protein . "\n";
print $result . "\n";
print $temp . "\n";


my $score = 0;
for(my $i=0; $i<length($result); $i++){
    if(substr($temp, $i, 1) eq substr($result, $i, 1)){
        $score++;
    }
}

print $score . "/" . length($result) . " (" . $score/length($result) . ")\n";
}
elsif($test==1){
    open(INFILE, "513_distribute/list.txt") || die("Could not open file!");
@filelist = <INFILE>;
close(INFILE);

my $dir = "513_distribute/";

    my $score1 = 0;
    my $score2 = 0;
    my $total = 0;

foreach my $file(@filelist){
    open(INFILE, $dir.$file) || die("Could not open file!");
    @info = <INFILE>;
    close(INFILE);

    chomp $info[0];
    chomp $info[1];
    chomp $info[5];

    @line1 = split(/,/, substr($info[0], 4));
    @line2 = split(/,/, substr($info[1], 5));
    @line3 = split(/,/, substr($info[5], 7));

    my $correct1 = 0;
    my $correct2 = 0;
    my $choice, $chance;
    my $sstruc = "";


    my $temp = 0;
    my $i = 0;
    $protein = "";
    foreach my $letter(@line1){
        $protein = $protein . $letter;
    }

    for(my $i=0; $i<length($protein); $i=$i+2){
        $choice = 0;
        $chance = $log_matrix[$aa{substr($protein, $i, 1)}*20 + $aa{substr($protein,
 $i+1, 1)}][0];
        for(my $j=1; $j<9; $j++){
            my $temp = $log_matrix[$aa{substr($protein, $i, 1)}*20 +
 $aa{substr($protein, $i+1, 1)}][$j];
            if($temp > $chance){
                }
            }
            $sstruc = $sstruc . $choice;
            $struc_chance = $struc_chance + $chance;
        }

        $temp = "";
        for(my $i=0; $i<length($sstruc); $i++){
            $temp =  $temp . $struc[substr($sstruc, $i, 1)];
        }
        $temp =~ tr/HEC/012/;
        for($i=0; $i<length($temp); $i++){
            my $check = 2;
            if(exists $input{$line2[$i]}){
                $check = $input{$line2[$i]};
            }
            if(substr($temp, $i, 1) == $check){
                $correct1++;
            }
             $check = 2;
            if(exists $input{$line3[$i]}){
                $check = $input{$line3[$i]};
            }
            if(substr($temp, $i, 1) == $check){
                $correct2++;
            }
        }

        $score1 = $score1 + $correct1;
        $score2 = $score2 + $correct2;
        $total = $total + $i;
        #last;

    }
        print $score1 . "/" . $total . " (" . $score1/$total . ")\n";
        print $score2 . "/" . $total . " (" . $score2/$total . ")\n";

    }
    elsif($test == 2){
        open(INFILE, "406_distribute//list.txt") || die("Could not open file!");
        @filelist = <INFILE>;
        close(INFILE);

    my $dir = "406_distribute/";

        my $score1 = 0;
        my $score2 = 0;
        my $total = 0;

    foreach my $file(@filelist){
        open(INFILE, $dir.$file) || die("Could not open file!");
        @info = <INFILE>;
        close(INFILE);

        chomp $info[0];
        chomp $info[scalar @info - 2];

        @line1 = split(/,/, substr($info[0], 20));
        @line2 = split(/,/, substr($info[scalar @info - 2], 5));



        my $correct1 = 0;
        my $correct2 = 0;
    my $choice, $chance;
    my $sstruc = "";


    my $temp = 0;
    my $i = 0;
    $protein = "";
    foreach my $letter(@line1){
        $protein = $protein . $letter;
    }

    for(my $i=0; $i<length($protein); $i=$i+2){
        $choice = 0;
        $chance = $log_matrix[$aa{substr($protein, $i, 1)}*20 + $aa{substr($protein,
 $i+1, 1)}][0];
        for(my $j=1; $j<9; $j++){
            my $temp = $log_matrix[$aa{substr($protein, $i, 1)}*20 +
 $aa{substr($protein, $i+1, 1)}][$j];
            if($temp > $chance){
                $choice = $j;
                $chance = $temp;
            }
        }
        $sstruc = $sstruc . $choice;
        $struc_chance = $struc_chance + $chance;
    }

    $temp = "";
    for(my $i=0; $i<length($sstruc); $i++){
        $temp =  $temp . $struc[substr($sstruc, $i, 1)];
    }
    $temp =~ tr/HEC/012/;
    for($i=0; $i<length($temp); $i++){
        my $check = 2;
        if(exists $input{$line2[$i]}){
            $check = $input{$line2[$i]};
        }
        if(substr($temp, $i, 1) == $check){
            $correct1++;
        }
    }

    $score1 = $score1 + $correct1;
    $total = $total + $i;
    #last;

}
    print $score1 . "/" . $total . " (" . $score1/$total . ")\n";

}


exit;
```

```
my @file, @matrix, @log_matrix;
my %aa, @struc, %input;
$aa{"A"} = 0; $aa{"a"} = 0;
$aa{"C"} = 1; $aa{"c"} = 1;
$aa{"D"} = 2; $aa{"d"} = 2;
$aa{"E"} = 3; $aa{"e"} = 3;
$aa{"F"} = 4; $aa{"f"} = 4;
$aa{"G"} = 5; $aa{"g"} = 5;
$aa{"H"} = 6; $aa{"h"} = 6;
$aa{"I"} = 7; $aa{"i"} = 7;
$aa{"K"} = 8; $aa{"k"} = 8;
$aa{"L"} = 9; $aa{"l"} = 9;
$aa{"M"} = 10;$aa{"m"} = 10;
$aa{"N"} = 11;$aa{"n"} = 11;
$aa{"P"} = 12;$aa{"p"} = 12;
$aa{"Q"} = 13;$aa{"q"} = 13;
$aa{"R"} = 14;$aa{"r"} = 14;
$aa{"S"} = 15;$aa{"s"} = 15;
$aa{"T"} = 16;$aa{"t"} = 16;
$aa{"V"} = 17;$aa{"v"} = 17;
$aa{"W"} = 18;$aa{"w"} = 18;
$aa{"Y"} = 19;$aa{"y"} = 19;
#$struc[0] = 'HH'; $struc[1] = 'HE'; $struc[2] = 'HC';
#$struc[3] = 'EH'; $struc[4] = 'EE'; $struc[5] = 'EC';
#$struc[6] = 'CH'; $struc[7] = 'CE'; $struc[8] = 'CC';
$struc[0] = 'H'; $struc[1] = 'E'; $struc[2] = 'C';
$input{"G"} = 0; $input{"H"} = 0; $input{"I"} = 0;
$input{"E"} = 1; $input{"B"} = 1;

if($file eq ""){
    $file="1.cgi";
}

$protein =
 "VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTF
ATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH";
my $result =
 "CCCCHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHCCCCCCCCCCC
HHHHHHHHHCCCCCCCHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHHHHHHHHHCCCC";


if(-e "p2matrix.txt"){
    open(INFILE, "p2matrix.txt") || die("Could not open file!");
    @file = <INFILE>;
    close(INFILE);

    for(my $i=0; $i<@file; $i++){
        chomp $file[$i];
        my @temp = split(/ /, $file[$i]);
        for(my $j=0; $j<@temp; $j++){
            $log_matrix[$i][$j] = $temp[$j];
        }
    }
}
else {
    open(INFILE, "2matrix.txt") || die("Could not open file!");
    @file = <INFILE>;
    close(INFILE);

    for(my $i=0; $i<@file; $i++){
        chomp $file[$i];
        my @temp = split(/ /, $file[$i]);
        for(my $j=0; $j<@temp; $j++){
            $matrix[$i][$j] = $temp[$j] + 1;
    my $temp;
    #converts log-odds variable into log-odds
    foreach my $row(@log_matrix){
        $temp = 0;
        foreach my $val(@$row){
            $temp += $val;
        }

        foreach my $val(@$row){
            $val = log(($val/$temp)/(1/9));
        }
    }


    open(OUTFILE, ">p2matrix.txt") || die("Could not open file!");
    for(my $i=0; $i<400; $i++){
        for(my $j=0; $j<9; $j++){
            print OUTFILE $log_matrix[$i][$j] . " ";
        }
        print OUTFILE "\n";
    }
    close(OUTFILE);
}



#for(my $i=0; $i<$#matrix; $i++){
#   for(my $j=0; $j<$#matrix; $j++){
#       print $matrix[$i][$j] . " ";
#   }
#   print "\n";
#}
#print "\n\n";



#open(OUTFILE, ">p2matrix.txt") || die("Could not open file!");
#for(my $i=0; $i<400; $i++){
#   for(my $j=0; $j<9; $j++){
#       print OUTFILE $log_matrix[$i][$j] . " ";
#   }
#   print OUTFILE "\n";
#}
#close(OUTFILE);

my $choice, $chance, $struc_chance = 0;
my $sstruc = "";

#   for(my $j=1; $j<9; $j++){
#       my $temp = $log_matrix[$aa{substr($protein, $i, 1)}*20 + $aa{substr($protein,
$i+1, 1)}][$j];
#       if($temp > $chance){
#           $choice = $j;
#           $chance = $temp;
#       }
#}

#open(OUTFILE, ">p2test.txt") || die("Could not open file!");
#$sstruc = '8';
#for(my $i=2; $i<length($protein); $i=$i+2){
#   $x = substr($sstruc, $i/2, 1) % 3 * 3;
#   $choice = $x;
#    print OUTFILE $protein[$i] . " " . $protein[$i+1] . " ";
#    print OUTFILE $chance;
#
#    for(my $j=$x+1; $j<$x+3; $j++){
#        my $temp = $log_matrix[$aa{substr($protein, $i, 1)}*20 + $aa{substr($protein,
$i+1, 1)}][$j];
#        print OUTFILE " " . $temp;
#        if($temp > $chance){
#            $choice = $j;
#            $chance = $temp;
#        }
#    }
#    print OUTFILE "\n";
#    $sstruc = $sstruc . $choice;
#    $struc_chance = $struc_chance + $chance;
#}
#close(OUTFILE);


print "Choose Test: \n";
print "0) HBB\n";
print "1) CB513(training set)\n";
print "2) CB406(testing set)\n\n";
print "Choice: ";
my $test = <STDIN>;

if($test == 0){
$choice = 0;
$chance = $log_matrix[$aa{substr($protein, 0, 1)}*20 + $aa{substr($protein, 1, 1)}][0];
#print $chance . "\n";
for(my $j=1; $j<9; $j++){
    my $temp = $log_matrix[$aa{substr($protein, 0, 1)}*20 + $aa{substr($protein, 1,
 1)}][$j];
    #print $temp . "\n";
    if($temp > $chance){
        $choice = $j;
        $chance = $temp;
    }
}
$sstruc = int($choice/3) . ($choice%3);


for(my $i=2; $i<length($protein); $i++){
    $x = substr($sstruc, $i-1, 1);
    $choice = 3*$x;
    $chance = $log_matrix[$aa{substr($protein, $i-1, 1)}*20 + $aa{substr($protein, $i,
 1)}][3*$x];

    for(my $j=3*$x+1; $j<3*$x+3; $j++){
        my $temp = $log_matrix[$aa{substr($protein, $i-1, 1)}*20 + $aa{substr($protein,
 $i, 1)}][$j];
        if($temp > $chance){
            $choice = $j;
            $chance = $temp;
        }
    }
    $sstruc = $sstruc . ($choice%3);
    $struc_chance = $struc_chance + $chance;
}



$temp = "";
for(my $i=0; $i<length($sstruc); $i++){
```

```
print $result . "\n";
print $temp . "\n";
print $struc_chance . "\n";
}
elsif($test==1){
    open(INFILE, "513_distribute/list.txt") || die("Could not open file!");
@filelist = <INFILE>;
close(INFILE);

my $dir = "513_distribute/";

    my $score1 = 0;
    my $score2 = 0;
    my $total = 0;

foreach my $file(@filelist){
    open(INFILE, $dir.$file) || die("Could not open file!");
    @info = <INFILE>;
    close(INFILE);

    chomp $info[0];
    chomp $info[1];
    chomp $info[5];

    @line1 = split(/,/, substr($info[0], 4));
    @line2 = split(/,/, substr($info[1], 5));
    @line3 = split(/,/, substr($info[5], 7));

    my $correct1 = 0;
    my $correct2 = 0;
    my $choice, $chance;
    my $sstruc = "";


    my $temp = 0;
    my $i = 0;
    $protein = "";
    foreach my $letter(@line1){
        $protein = $protein . $letter;
    }

$choice = 0;
$chance = $log_matrix[$aa{substr($protein, 0, 1)}*20 + $aa{substr($protein, 1, 1)}][0];
#print $chance . "\n";
for(my $j=1; $j<9; $j++){
    my $temp = $log_matrix[$aa{substr($protein, 0, 1)}*20 + $aa{substr($protein, 1,
 1)}][$j];
    #print $temp . "\n";
    if($temp > $chance){
        $choice = $j;
        $chance = $temp;
    }
}
$sstruc = int($choice/3) . ($choice%3);


for(my $i=2; $i<length($protein); $i++){
    $x = substr($sstruc, $i-1, 1);
    $choice = 3*$x;
    $chance = $log_matrix[$aa{substr($protein, $i-1, 1)}*20 + $aa{substr($protein, $i,
 1)}][3*$x];

    for(my $j=3*$x+1; $j<3*$x+3; $j++){
        my $temp = $log_matrix[$aa{substr($protein, $i-1, 1)}*20 + $aa{substr($protein,
            $chance = $temp;
        }
    }
    $sstruc = $sstruc . ($choice%3);
    $struc_chance = $struc_chance + $chance;
}

    for($i=0; $i<length($sstruc); $i++){
        my $check = 2;
        if(exists $input{$line2[$i]}){
            $check = $input{$line2[$i]};
        }
        if(substr($sstruc, $i, 1) == $check){
            $correct1++;
        }
         $check = 2;
        if(exists $input{$line3[$i]}){
            $check = $input{$line3[$i]};
        }
        if(substr($sstruc, $i, 1) == $check){
            $correct2++;
        }
    }

    $score1 = $score1 + $correct1;
    $score2 = $score2 + $correct2;
    $total = $total + $i;
    #last;

}
    print $score1 . "/" . $total . " (" . $score1/$total . ")\n";
    print $score2 . "/" . $total . " (" . $score2/$total . ")\n";
}
elsif($test == 2){
    open(INFILE, "406_distribute//list.txt") || die("Could not open file!");
    @filelist = <INFILE>;
    close(INFILE);

my $dir = "406_distribute/";

    my $score1 = 0;
    my $score2 = 0;
    my $total = 0;

foreach my $file(@filelist){
    open(INFILE, $dir.$file) || die("Could not open file!");
    @info = <INFILE>;
    close(INFILE);

    chomp $info[0];
    chomp $info[scalar @info - 2];

    @line1 = split(/,/, substr($info[0], 20));
    @line2 = split(/,/, substr($info[scalar @info - 2], 5));



    my $correct1 = 0;
    my $correct2 = 0;
    my $choice, $chance;
    my $sstruc = "";


    my $temp = 0;
    my $i = 0;
    $protein = "";
    foreach my $letter(@line1){
        $protein = $protein . $letter;
    }

$choice = 0;
$chance = $log_matrix[$aa{substr($protein, 0, 1)}*20 + $aa{substr($protein, 1, 1)}][0];
#print $chance . "\n";
for(my $j=1; $j<9; $j++){
    my $temp = $log_matrix[$aa{substr($protein, 0, 1)}*20 + $aa{substr($protein, 1,
 1)}][$j];
    #print $temp . "\n";
    if($temp > $chance){
        $choice = $j;
        $chance = $temp;
    }
}
$sstruc = int($choice/3) . ($choice%3);


for(my $i=2; $i<length($protein); $i++){
    $x = substr($sstruc, $i-1, 1);
    $choice = 3*$x;
    $chance = $log_matrix[$aa{substr($protein, $i-1, 1)}*20 + $aa{substr($protein, $i,
 1)}][3*$x];

    for(my $j=3*$x+1; $j<3*$x+3; $j++){
        my $temp = $log_matrix[$aa{substr($protein, $i-1, 1)}*20 + $aa{substr($protein,
 $i, 1)}][$j];
        if($temp > $chance){
            $choice = $j;
            $chance = $temp;
        }
    }
    $sstruc = $sstruc . ($choice%3);
    $struc_chance = $struc_chance + $chance;
}

    for($i=0; $i<length($sstruc); $i++){
        my $check = 2;
        if(exists $input{$line2[$i]}){
            $check = $input{$line2[$i]};
        }
        if(substr($sstruc, $i, 1) == $check){
            $correct1++;
        }
         $check = 2;
        if(exists $input{$line3[$i]}){
            $check = $input{$line3[$i]};
        }
        if(substr($sstruc, $i, 1) == $check){
            $correct2++;
        }
    }

    $score1 = $score1 + $correct1;
    $total = $total + $i;
    #last;

}
    print $score1 . "/" . $total . " (" . $score1/$total . ")\n";

}
```

```
exit;
```

```perl
my @filelist, @matrix, @sss_matrix, @start_stop_matrix;
my %aa, %struc, %tol;
$aa{"A"} = 0; $aa{"a"} = 0;
$aa{"C"} = 1; $aa{"c"} = 1;
$aa{"D"} = 2; $aa{"d"} = 2;
$aa{"E"} = 3; $aa{"e"} = 3;
$aa{"F"} = 4; $aa{"f"} = 4;
$aa{"G"} = 5; $aa{"g"} = 5;
$aa{"H"} = 6; $aa{"h"} = 6;
$aa{"I"} = 7; $aa{"i"} = 7;
$aa{"K"} = 8; $aa{"k"} = 8;
$aa{"L"} = 9; $aa{"l"} = 9;
$aa{"M"} = 10;$aa{"m"} = 10;
$aa{"N"} = 11;$aa{"n"} = 11;
$aa{"P"} = 12;$aa{"p"} = 12;
$aa{"Q"} = 13;$aa{"q"} = 13;
$aa{"R"} = 14;$aa{"r"} = 14;
$aa{"S"} = 15;$aa{"s"} = 15;
$aa{"T"} = 16;$aa{"t"} = 16;
$aa{"V"} = 17;$aa{"v"} = 17;
$aa{"W"} = 18;$aa{"w"} = 18;
$aa{"Y"} = 19;$aa{"y"} = 19;


#helix (G, H and I), strand (E and B) and loop (all others).
$struc{"G"} = 0;$struc{"H"} = 0;$struc{"I"} = 0;
$struc{"E"} = 1;$struc{"B"} = 1;
$protein =
 "VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTF
ATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH";
my $result =
 "CCCCHHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHCCCCCCCCCC
HHHHHHHHHCCCCCCHHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHHHHHHHHHHHHCCCC";

#$protein = "CHRLTTVCPTSKPQTQGLAKDAWEIPRESLRLEVKLGQGCFGEVWMGTWNGTTRVAIKTL";

open(INFILE, "1pmatrix.txt") || die("Could not open file!");
@file = <INFILE>;
close(INFILE);

for(my $i=0; $i<@file; $i++){
    chomp $file[$i];
    my @temp = split(/ /, $file[$i]);
    for(my $j=0; $j<@temp; $j++){
        $matrix[$i][$j] = $temp[$j];
    }
}

open(INFILE, "1zzzmatrix.txt") || die("Could not open file!");
@file = <INFILE>;
close(INFILE);

for(my $i=0; $i<@file; $i++){
    chomp $file[$i];
    my @temp = split(/ /, $file[$i]);
    for(my $j=0; $j<@temp; $j++){
        $sss_matrix[$i][$j] = $temp[$j];
    }
}

print "Choose Test: \n";
print "0) HBB\n";
print "1) CB513(training set)\n";
print "2) CB406(testing set)\n\n";
```

```
    $score = 0;
    my $choice = 0;
    my $chance = $sss_matrix[$aa{substr($protein, 0, 1)}][1];
    my $temp = $sss_matrix[$aa{substr($protein, 0, 1)}][4];
    if($temp > $chance){
        $choice = 1;
        $chance = $temp;
    }

    $temp = $sss_matrix[$aa{substr($protein, 0, 1)}][7];
    if($temp > $chance){
        $choice = 2;
        $chance = $temp;
    }

    $sstruc = $sstruc . $choice;
    $prev = $choice;
    for(my $i=1; $i<length($protein); $i++){
        $choice = $prev;
        $chance = $matrix[$aa{substr($protein, $i, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i, 1)}][$prev*3+1];

        $stop = $matrix[$aa{substr($protein, $i-1, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i-1, 1)}][$prev*3+2];
        $start1 = $matrix[$aa{substr($protein, $i, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i, 1)}][($prev+1)%3*3];
        $start2 = $matrix[$aa{substr($protein, $i, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i, 1)}][($prev+2)%3*3];
        if($start+$stop > $chance){
            $chance = $start1+$stop;
            $choice = ($prev+1)%3;
        }

        if($start2+$stop > $chance){
            $chance = $start2+$stop;
            $choice = ($prev+2)%3;
        }

    #print substr($protein, $i, 1) . "  stay:" . $chance . "  stop:" . $stop;
    #print "  start1:" . $start1 . "  start2:" . $start2 . "\n";

    $sstruc = $sstruc . $choice;
    $prev = $choice;
    #print $sstruc . "\n";
}
    $sstruc =~ tr/012/HEC/;
    for(my $i=0; $i<length($protein); $i++){
        if(substr($sstruc, $i, 1) eq substr($result, $i, 1)){
            $correct++;
        }
    }

    $score = $score + $correct;
    $total = $total + length($protein);

    print $protein . "\n";
    print $result . "\n";

    print $sstruc . "\n";

    print $score . "/" . $total . " (" . $score/$total . ")\n";


}
```

```
my $dir = "513_distribute/";

my $prev = $choice;

    my $score = 0;
    my $total = 0;

foreach my $file(@filelist){
    open(INFILE, $dir.$file) || die("Could not open file!");
    @info = <INFILE>;
    close(INFILE);

    chomp $info[0];
    chomp $info[1];
    chomp $info[5];

    @line1 = split(/,/, substr($info[0], 4));
    @line2 = split(/,/, substr($info[1], 5));
    @line3 = split(/,/, substr($info[5], 7));

    my $protein = "";

    my $correct = 0;
    my $choice, $struc_chance = 0;
    my $chance, $sstruc;

    my $temp = 0;
    my $i = 0;
    foreach my $letter(@line1){
        $protein = $protein . $letter;
    }

    $sstruc = "";


my $choice = 0;
my $chance = $sss_matrix[$aa{substr($protein, 0, 1)}][1];
my $temp = $sss_matrix[$aa{substr($protein, 0, 1)}][4];
if($temp > $chance){
    $choice = 1;
    $chance = $temp;
}

$temp = $sss_matrix[$aa{substr($protein, 0, 1)}][7];
if($temp > $chance){
    $choice = 2;
    $chance = $temp;
}
#$choice = 2;
$sstruc = $sstruc . $choice;
$prev = $choice;
for(my $i=1; $i<length($protein); $i++){
    $choice = $prev;
    $chance = $matrix[$aa{substr($protein, $i, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i, 1)}][$prev*3+1];

    $stop = $matrix[$aa{substr($protein, $i-1, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i-1, 1)}][$prev*3+2];
    $start1 = $matrix[$aa{substr($protein, $i, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i, 1)}][($prev+1)%3*3];
    $start2 = $matrix[$aa{substr($protein, $i, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i, 1)}][($prev+2)%3*3];
    if($start1 > $chance || $stop > $chance){
```

```
    if($start2 > $chance){
        $chance = $start2+$stop;
        $choice = ($prev+2)%3;
    }

    #print substr($protein, $i, 1) . "  stay:" . $chance . "  stop:" . $stop;
    #print "  start1:" . $start1 . "  start2:" . $start2 . "\n";

    $sstruc = $sstruc . $choice;
    $prev = $choice;
    #print $sstruc . "\n";
}

    for(my $i=0; $i<length($protein); $i++){
        my $check = 2;
        if(exists $struc{$line2[$i]}){
            $check = $struc{$line2[$i]};
        }
        if(substr($sstruc, $i, 1) eq $check){
            $correct++;
        }
    }

    $score = $score + $correct;
    $total = $total + length($protein);

    #last;

}
    print $score . "/" . $total . " (" . $score/$total . ")\n";
    #print $protein . "\n";
    #print $sstruc;
}
elsif($test == 2){
    open(INFILE, "406_distribute/list.txt") || die("Could not open file!");
    @filelist = <INFILE>;
    close(INFILE);

    my $dir = "406_distribute/";

    my $score = 0;
    my $total = 0;

    foreach my $file(@filelist){
        open(INFILE, $dir.$file) || die("Could not open file!");
        @info = <INFILE>;
        close(INFILE);

    chomp $info[0];
    chomp $info[scalar @info - 2];

    @line1 = split(/,/, substr($info[0], 20));
    @line2 = split(/,/, substr($info[scalar @info - 2], 5));

    my $protein = "";

    my $correct = 0;
    my $choice, $struc_chance = 0;
    my $chance, $sstruc;

    my $temp = 0;
    my $i = 0;
    foreach my $letter(@line1){
        $protein = $protein . $letter;
    }

    $sstruc = "";


my $choice = 0;
my $chance = $sss_matrix[$aa{substr($protein, 0, 1)}][1];
my $temp = $sss_matrix[$aa{substr($protein, 0, 1)}][4];
if($temp > $chance){
    $choice = 1;
    $chance = $temp;
}

$temp = $sss_matrix[$aa{substr($protein, 0, 1)}][7];
if($temp > $chance){
    $choice = 2;
    $chance = $temp;
}
#$choice = 2;
$sstruc = $sstruc . $choice;
$prev = $choice;
for(my $i=1; $i<length($protein); $i++){
    $choice = $prev;
    $chance = $matrix[$aa{substr($protein, $i, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i, 1)}][$prev*3+1];

    $stop = $matrix[$aa{substr($protein, $i-1, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i-1, 1)}][$prev*3+2];
    $start1 = $matrix[$aa{substr($protein, $i, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i, 1)}][($prev+1)%3*3];
    $start2 = $matrix[$aa{substr($protein, $i, 1)}][$prev] *
 $sss_matrix[$aa{substr($protein, $i, 1)}][($prev+2)%3*3];
    if($start1 > $chance || $stop > $chance){
        $chance = $start1;#+$stop;
        $choice = ($prev+1)%3;
    }

    if($start2 > $chance){
        $chance = $start2+$stop;
        $choice = ($prev+2)%3;
    }

    #print substr($protein, $i, 1) . "  stay:" . $chance . "  stop:" . $stop;
    #print "  start1:" . $start1 . "  start2:" . $start2 . "\n";

    $sstruc = $sstruc . $choice;
    $prev = $choice;
    #print $sstruc . "\n";
}

    for(my $i=0; $i<length($protein); $i++){
        my $check = 2;
        if(exists $struc{$line2[$i]}){
            $check = $struc{$line2[$i]};
        }
        #print $line2[$i];
        if(substr($sstruc, $i, 1) eq $check){
            $correct++;
        }
    }
    #print "\n";

    $score = $score + $correct;
    $total = $total + length($protein);
    print $score . "/" . $total . " (" . $score/$total . ")\n";
}




exit;
```

```
my @file, @matrix, @log_matrix, @tmatrix, @log_tmatrix, @imatrix, @log_imatrix;
my %aa, @struc, %input;
$aa{"A"} = 0; $aa{"a"} = 0;
$aa{"C"} = 1; $aa{"c"} = 1;
$aa{"D"} = 2; $aa{"d"} = 2;
$aa{"E"} = 3; $aa{"e"} = 3;
$aa{"F"} = 4; $aa{"f"} = 4;
$aa{"G"} = 5; $aa{"g"} = 5;
$aa{"H"} = 6; $aa{"h"} = 6;
$aa{"I"} = 7; $aa{"i"} = 7;
$aa{"K"} = 8; $aa{"k"} = 8;
$aa{"L"} = 9; $aa{"l"} = 9;
$aa{"M"} = 10;$aa{"m"} = 10;
$aa{"N"} = 11;$aa{"n"} = 11;
$aa{"P"} = 12;$aa{"p"} = 12;
$aa{"Q"} = 13;$aa{"q"} = 13;
$aa{"R"} = 14;$aa{"r"} = 14;
$aa{"S"} = 15;$aa{"s"} = 15;
$aa{"T"} = 16;$aa{"t"} = 16;
$aa{"V"} = 17;$aa{"v"} = 17;
$aa{"W"} = 18;$aa{"w"} = 18;
$aa{"Y"} = 19;$aa{"y"} = 19;
$struc[0] = 'H'; $struc[1] = 'E'; $struc[2] = 'C';
$input{"G"} = 0; $input{"H"} = 0; $input{"I"} = 0;
$input{"E"} = 1; $input{"B"} = 1;


if($file eq ""){
    $file="1.cgi";
}

$protein =
 "VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTF
ATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH";
my $result =
 "CCCCHHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHHCCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHCCCCCCCCCC
HHHHHHHHHCCCCCCCHHHHHHHHHHHHHHHHHHHCCCCHHHHHHHHHHHHHHHHHHHHCCCC";
#$protein = "CHRLTTVCPTSKPQTQGLAKDAWEIPRESLRLEVKLGQGCFGEVWMGTWNGTTRVAIKTL";
open(INFILE, "1pmatrix.txt") || die("Could not open file!");
@file = <INFILE>;
close(INFILE);

for(my $i=0; $i<@file; $i++){
    chomp $file[$i];
    my @temp = split(/ /, $file[$i]);
    for(my $j=0; $j<@temp; $j++){
        $matrix[$i][$j] = $temp[$j];
        $log_matrix[$i][$j] = $temp[$j];
    }

}



print "Choose Test: \n";
print "0) HBB\n";
print "1) CB513(training set)\n";
print "2) CB406(testing set)\n\n";
print "Choice: ";
my $test = <STDIN>;

if($test == 0){

my $choice, @chance;
```

```
for(my $i=0; $i<length($protein); $i++){
    $chance[0] = $matrix[$aa{substr($protein, $i, 1)}][0];
    $chance[1] = $matrix[$aa{substr($protein, $i, 1)}][1];
    $chance[2] = $matrix[$aa{substr($protein, $i, 1)}][2];
    $choice = 0;
    for(my $j=1; $j<3; $j++){
            if($chance[$j] > $chance[$choice]){
                $choice = $j;
            }
        }
        $sstruc[0] = $sstruc[0] . $choice;
    for(my $k=1; $k<8; $k++){
        $choice = 0;
        $chance[0] = $chance[0] + $matrix[$aa{substr($protein, $i-$k, 1)}][0] +
 $matrix[$aa{substr($protein, $i+$k, 1)}][0];
        $chance[1] = $chance[1] + $matrix[$aa{substr($protein, $i-$k, 1)}][1] +
 $matrix[$aa{substr($protein, $i+$k, 1)}][1];
        $chance[2] = $chance[2] + $matrix[$aa{substr($protein, $i-$k, 1)}][2] +
 $matrix[$aa{substr($protein, $i+$k, 1)}][2];

        for(my $j=1; $j<3; $j++){
            if($chance[$j] > $chance[$choice]){
                $choice = $j;
            }
        }
        $sstruc[$k] = $sstruc[$k] . $choice;
    }

}

my $con = "";
for(my $i=0; $i<length($protein); $i++){
    my $add = 0;
    my @count;
    $count[0] = 0;
    $count[1] = 0;
    $count[2] = 0;
    for(my $k=1; $k<3; $k++){
        $count[substr($sstruc[$k], $i, 1)]++;
    }

    if($count[1]>$count[$add]){
        $add = 1;
    }
    if($count[2]>$count[$add]){
        $add = 2;
    }
    $con = $con . $add;

}

print $protein . "\n";
print $result . "\n";

my $after_con = $con;
$con =~ tr/012/HEC/;
print $con . "\n";

for(my $i=1; $i<length($after_con)-3; $i++){
    my $count = 0;

    my $prev = substr($after_con, $i-1, 1);
    my $letter = substr($after_con, $i, 1);
```

```
                $count++;
            }
        }

        if($count<2 && $prev != $letter){
            $after_con = substr($after_con, 0, $i) . substr($after_con, $i-1, 1) .
     substr($after_con, $i+1, length($after_con)-$i);
        }


    }


    my $score = 0;
    for(my $i=0; $i<length($result); $i++){
        if($struc[substr($after_con, $i, 1)] eq substr($result, $i, 1)){
            $score++;
        }
    }
    $after_con =~ tr/012/HEC/;
    print $after_con . "\n";

    print $score . "/" . length($result) . " (" . $score/length($result) . ")\n";
    }
    elsif($test == 1){
        open(INFILE, "513_distribute/list.txt") || die("Could not open file!");
    @filelist = <INFILE>;
    close(INFILE);

    my $dir = "513_distribute/";

        my $score1 = 0;
        my $score2 = 0;
        my $total = 0;

        my $low = 1;
        my $high = 0;

    foreach my $file(@filelist){
        open(INFILE, $dir.$file) || die("Could not open file!");
        @info = <INFILE>;
        close(INFILE);

        chomp $info[0];
        chomp $info[1];
        chomp $info[5];

        @line1 = split(/,/, substr($info[0], 4));
        @line2 = split(/,/, substr($info[1], 5));
        @line3 = split(/,/, substr($info[5], 7));

        my $correct1 = 0;
        my $correct2 = 0;
        my $choice, $chance;
        my $sstruc = "";


        my $temp = 0;
        my $i = 0;
        $protein = "";
        foreach my $letter(@line1){
            $protein = $protein . $letter;
        }
    my $choice, @chance;
```

```
for(my $i=0; $i<8; $i++){
    $sstruc[$i] = "";
}

for(my $i=0; $i<length($protein); $i++){
    $chance[0] = $matrix[$aa{substr($protein, $i, 1)}][0];
    $chance[1] = $matrix[$aa{substr($protein, $i, 1)}][1];
    $chance[2] = $matrix[$aa{substr($protein, $i, 1)}][2];
    $choice = 0;
    for(my $j=1; $j<3; $j++){
            if($chance[$j] > $chance[$choice]){
                $choice = $j;
            }
        }
        $sstruc[0] = $sstruc[0] . $choice;
    for(my $k=1; $k<8; $k++){
        $choice = 0;
        $chance[0] = $chance[0] + $matrix[$aa{substr($protein, $i-$k, 1)}][0] +
 $matrix[$aa{substr($protein, $i+$k, 1)}][0];
        $chance[1] = $chance[1] + $matrix[$aa{substr($protein, $i-$k, 1)}][1] +
 $matrix[$aa{substr($protein, $i+$k, 1)}][1];
        $chance[2] = $chance[2] + $matrix[$aa{substr($protein, $i-$k, 1)}][2] +
 $matrix[$aa{substr($protein, $i+$k, 1)}][2];

        for(my $j=1; $j<3; $j++){
            if($chance[$j] > $chance[$choice]){
                $choice = $j;
            }
        }
        $sstruc[$k] = $sstruc[$k] . $choice;
    }

}

my $con = "";
for(my $i=0; $i<length($protein); $i++){
    my $add = 0;
    my @count;
    $count[0] = 0;
    $count[1] = 0;
    $count[2] = 0;
    for(my $k=1; $k<3; $k++){
        $count[substr($sstruc[$k], $i, 1)]++;
    }

    if($count[1]>$count[$add]){
        $add = 1;
    }
    if($count[2]>$count[$add]){
        $add = 2;
    }
    $con = $con . $add;

}



my $after_con = $con;


for(my $i=1; $i<length($after_con)-3; $i++){
    my $count = 0;

    my $prev = substr($after_con, $i-1, 1);
```

```
            if(substr($after_con, $i, 1) == substr($after_con, $i+$j, 1)){
                $count++;
            }
        }

        if($count<2 && $prev != $letter){
            $after_con = substr($after_con, 0, $i) . substr($after_con, $i-1, 1) .
   substr($after_con, $i+1, length($after_con)-$i);
        }


    }


        for(my $i=0; $i<length($protein); $i++){
            my $check = 2;
            if(exists $input{$line2[$i]}){
                $check = $input{$line2[$i]};
                #print $struc{$line2[$i]};
            }
            if(substr($after_con, $i, 1) eq $check){
                $correct1++;
            }

            $check = 2;
            if(exists $input{$line3[$i]}){
                $check = $input{$line3[$i]};
            }
            if(substr($after_con, $i, 1) eq $check){
                $correct2++;
            }
        }
        $score1 = $score1 + $correct1;
        $score2 = $score2 + $correct2;
        $total = $total + length($protein);

        if($correct1/length($protein)<$low){
            $low = $correct1/length($protein);
        }
        if($correct1/length($protein)>$high){
            $high = $correct1/length($protein);
        }
            #if($correct1/length($protein)>0.9){
            #print $correct1/length($protein) . "\n";
            #print $protein . "\n";
            #for(my $i=0; $i<length($protein); $i++){
            #   print $line2[$i];
            #}
            #$after_con =~ tr/012/HEC/;
            #print "\n" . $after_con . "\n";
        #}
    }
    print $score1 . "/" . $total . " (" . $score1/$total . ")\n";
    #print "low: " . $low . "  high: " . $high . "\n";
    }
    elsif($test==2){
        open(INFILE, "406_distribute//list.txt") || die("Could not open file!");
        @filelist = <INFILE>;
        close(INFILE);

    my $dir = "406_distribute/";

        my $score1 = 0;
        my $score2 = 0;
    my $low = 1;
    my $high = 0;

foreach my $file(@filelist){
    open(INFILE, $dir.$file) || die("Could not open file!");
    @info = <INFILE>;
    close(INFILE);

    chomp $info[0];
    chomp $info[scalar @info - 2];

    @line1 = split(/,/, substr($info[0], 20));
    @line2 = split(/,/, substr($info[scalar @info - 2], 5));

    my $correct1 = 0;
    my $correct2 = 0;
    my $choice, $chance;
    my $sstruc = "";


    my $temp = 0;
    my $i = 0;
    $protein = "";
    foreach my $letter(@line1){
        $protein = $protein . $letter;
    }
my $choice, @chance;
my @sstruc;
for(my $i=0; $i<8; $i++){
    $sstruc[$i] = "";
}

for(my $i=0; $i<length($protein); $i++){
    $chance[0] = $matrix[$aa{substr($protein, $i, 1)}][0];
    $chance[1] = $matrix[$aa{substr($protein, $i, 1)}][1];
    $chance[2] = $matrix[$aa{substr($protein, $i, 1)}][2];
    $choice = 0;
    for(my $j=1; $j<3; $j++){
            if($chance[$j] > $chance[$choice]){
                $choice = $j;
            }
        }
        $sstruc[0] = $sstruc[0] . $choice;
    for(my $k=1; $k<8; $k++){
        $choice = 0;
        $chance[0] = $chance[0] + $matrix[$aa{substr($protein, $i-$k, 1)}][0] +
 $matrix[$aa{substr($protein, $i+$k, 1)}][0];
        $chance[1] = $chance[1] + $matrix[$aa{substr($protein, $i-$k, 1)}][1] +
 $matrix[$aa{substr($protein, $i+$k, 1)}][1];
        $chance[2] = $chance[2] + $matrix[$aa{substr($protein, $i-$k, 1)}][2] +
 $matrix[$aa{substr($protein, $i+$k, 1)}][2];

        for(my $j=1; $j<3; $j++){
            if($chance[$j] > $chance[$choice]){
                $choice = $j;
            }
        }
        $sstruc[$k] = $sstruc[$k] . $choice;
    }

}

my $con = "";
```

```
        $count[0] = 0;
        $count[1] = 0;
        $count[2] = 0;
        for(my $k=1; $k<3; $k++){
            $count[substr($sstruc[$k], $i, 1)]++;
        }

        if($count[1]>$count[$add]){
            $add = 1;
        }
        if($count[2]>$count[$add]){
            $add = 2;
        }
        $con = $con . $add;

    }



    my $after_con = $con;


    for(my $i=1; $i<length($after_con)-3; $i++){
        my $count = 0;

        my $prev = substr($after_con, $i-1, 1);
        my $letter = substr($after_con, $i, 1);

        for(my $j=1; $j<5;$j++){
            if(substr($after_con, $i, 1) == substr($after_con, $i+$j, 1)){
                $count++;
            }
        }

        if($count<2 && $prev != $letter){
            $after_con = substr($after_con, 0, $i) . substr($after_con, $i-1, 1) .
     substr($after_con, $i+1, length($after_con)-$i);
        }


    }


        for(my $i=0; $i<length($protein); $i++){
            my $check = 2;
            if(exists $input{$line2[$i]}){
                $check = $input{$line2[$i]};
                #print $struc{$line2[$i]};
            }
            if(substr($after_con, $i, 1) eq $check){
                $correct1++;
            }

            $check = 2;
            if(exists $input{$line3[$i]}){
                $check = $input{$line3[$i]};
            }
            if(substr($after_con, $i, 1) eq $check){
                $correct2++;
            }
        }
        $score1 = $score1 + $correct1;
        $score2 = $score2 + $correct2;
        $total = $total + length($protein);
    if($correct1/length($protein)<$low){
        $low = $correct1/length($protein);
    }
    if($correct1/length($protein)>$high){
        $high = $correct1/length($protein);

    }

    #if($correct1/length($protein)>0.8){
    #   print $file . "\n";
    #   print $correct1/length($protein) . "\n";
    #   print $protein . "\n";
    #   for(my $i=0; $i<length($protein); $i++){
    #       print $line2[$i];
    #   }
    #   $after_con =~ tr/012/HEC/;
    #   print "\n" . $after_con . "\n";
    #}

}
print $score1 . "/" . $total . " (" . $score1/$total . ")\n";

#print "low: " . $low . "  high: " . $high . "\n";

}


exit;
```