Spring 2012

A Dynamic Programming Approach to Sequence Alignments

Jeffery Allen
Carroll College, Helena, MT

SIGNATURE PAGE

This thesis for honors recognition has been approved for the

Department of Mathematics, Engineering, and Computer Science

Director — Date: 3/28/12

Reader — Date: 3/28/12

Reader — Date: 3-28-12

A Dynamic Programming Approach to Sequence Alignments

Jeffery M. Allen
Department of Mathematics, Engineering, and Computer Science
Carroll College, Helena, MT

March 28, 2012

Abstract

Using a created a program in MatLab, several amino acid sequences can be aligned with one another so they can be examined for biological significance. Sequence alignments are used to determine similarities in proteins and from this information, make inferences on protein functionality. The program accomplishes this by setting a cost for gaps in a sequence or substitution of one amino acid for another. A dynamic programming algorithm called the Needleman & Wunsch Algorithm is used to find several possible global pairwise alignments. These pairwise alignments are used to form a multiple sequence alignment, which compares a number of like sequences together to look for highly conserved, and therefore possibly biologically important, sections.

A Dynamic Programming Approach to Sequence Alignments

Contents

1 Introduction 1

2 The Dot Plot 6

2.1 An Example Dot Plot 7

3 Needleman & Wunsch Algorithm 9

3.1 The Gap Cost 13

3.2 Gap Extensions 16

3.3 Real/Long Sequences 18

4 Multiple Sequence Alignment Program 21

5 Future Improvements/Conclusions 26

References 28

Appendix A: BLOSUM40 Table[4] 30

Appendix B: Needleman & Wunsch Algorithm Code 31

Appendix C: Multiple Sequence Alignment Code 43

1 INTRODUCTION

All life on this planet uses DNA or RNA as the blueprints for existence. There are two major categories of organisms. These are prokaryotes and eukaryotes. Prokaryotes are small single cell organisms such as bacteria, while eukaryotes tend to be multi-cellular and far more complex, such as humans. Although prokaryotes' genes can be made up of either DNA or RNA, eukaryotes use double-stranded DNA as the primary storage location for genetic data. DNA is made up of four different nucleic acid bases called adenine (A), thymine (T), guanine (G), and cytosine (C). These bases pair with one another (A with T and G with C) in such a way that one strand of DNA will attach to its complement to create a helix.

The human genome encodes an estimated 25000 proteins that perform the functions that keep us living. These proteins are created from the data stored in DNA through what is known as the Central Dogma of Molecular Biology: DNA is transcribed into RNA which is translated into proteins. The first process is known as transcription and the second process is translation. Transcription starts with unzipping the helical structure of DNA with a protein called Helicase. Then RNA-polymerase attaches to the DNA at the start of a gene known as the promoter region. RNA-polymerase travels along the single strand of DNA pairing each base with its complement. In RNA thymine is replaced with uracil (U), but the pairings remain the same. When RNA-polymerase finishes, the final product is known as messenger RNA (mRNA). In eukaryotes mRNA must be processed before moving on to translation by adding a methlyated cap and a poly-A tail. The methylated cap is a modified version of the amino acid guanine that prevents RNases from degrading the mRNA and signals a ribosome to begin translation. The poly-A tail is a string of adenines that keeps the mRNA from being degraded as well as provides structural support. In order to translate mRNA, sometimes large non-coding regions need to be removed. These are known as introns and the remaining parts are known as exons.

Translation starts when a ribosome attaches to the mRNA. The ribosome reads down the mRNA until it reaches a start codon. A codon is a set of three nucleic acids bases that code to either start translation, stop translation, or add a specific amino acid. Since there are four bases, there are $4^3 = 64$ possible combinations. However, there are only 20 amino acids, which means several codons represent one amino acid. The codon that indicated when the ribosome needs to

start coding is AUG and codes for the amino acid methionine. When the ribosome reaches the AUG site, a tRNA carrying the methionine amino acid enters the ribosome and binds to the AUG. At the "top" of tRNA, there is a free OH group that binds to an amino acid, and at the "bottom" there in an anti-codon. This anti-codon is the complement to the codon that codes for the amino acid, in this case UAC. There are various tRNAs for each codon and amino acid. This is how the correct tRNA-amino acid complex is selected when reading the mRNA.

The ribosome continues down the mRNA until it reaches the next three bases. The appropriate tRNA carrying the correct amino acid enters the ribosome. The ribosome then connects the two amino acids through a peptide bond, moves to the next codon, connects the next amino acid, and the process continues until a stop codon is reached, which is UAA, UAG, or UGA. The string of amino acids forms the primary structure of the protein. Through interaction with itself and its surroundings, the protein folds into its secondary and tertiary structures. Finally the protein joins to other proteins to be functional, if need be, forming the quaternary structure.

DNA is a useful storage device because it is relatively stable, and the four bases provide enough variability to encode the thousands of proteins in a condensed form. It is rather like binary but uses 0-3 rather than 0 and 1. Similar to binary, if one of the bases is wrong, the data is not read properly and, in the case of DNA, the whole protein can be malformed. This can happen several ways, but there are two important ways that can create long lasting effects. These are point mutations and insertions/deletions of nucleic acids.

Point mutation happens when one nucleotide is replaced with another. This mutation is less likely to cause a problem since there is redundancy in the codons. If the last base in a codon is switched, it is possible that the new codon will code for the same amino acid. This is known as a silent mutation. The more devastating mutations are the insertions and deletions. Because the string of nucleic acids is read in sequence and every group of three bases is read as a single unit, shifting the process by even one base could potentially change every codon and therefore create an entire new protein. If either mutation interrupts or creates a start or stop codon, the resulting protein will be the wrong length and most likely not work properly.

Each mutation has a chance of creating a protein that works more or less efficiently. When not influenced by other pressures, the better working proteins are an advantage to their owner and

the organism will thrive and spread its mutation. Likewise, if the protein works less efficiently, the owner will probably die and thus remove that mutation from the gene pool. Over millions of years, evolutionary forces drive species of organisms apart until they are no longer the same species. The interesting fact is that distantly related species sometimes share sections of DNA that code for nearly the same proteins. Because of this, it is possible to look at a protein with an unknown function and compare it with proteins with known functions and hypothesize the original protein's function. This is done through the method of sequence alignments as well as some other more complicated methods.

Sequence alignments look at the primary structure of a protein and compare it to one or multiple other primary structures. Once the primary structure is created, the amino acids interact with each other to form the secondary and tertiary structures. It is the tertiary structure that ultimately determines functionality. Similar proteins with the same function could have the same tertiary structure but different primary structures. The goal of sequence alignments is to compare proteins and look for a structural motif, highly conserved regions, and highly variable regions. These data help in determining the function of a protein.

The two major categories of alignments are called pairwise and multiple sequence alignments. A pairwise alignment happens when only two sequences are being compared, and multiple sequence alignments compare more than two sequences. Some common methods of sequence alignments include dot plots, dynamic programs, heuristic programs, and, in the case of short sequences, by hand. Of the multiple ways to align sequences, there are a few that only work for pairwise alignments. One of these is known as the dot plot. A dot plot is a simple graphical representation of common regions between two sequences.[5] Other pairwise exclusive methods include rather complex dynamic programs that become infeasible with more than two sequences and the manual "by hand" method, although, one could argue that this method works for more than two sequences depending on the patience of the person aligning them.

Dynamic programs tend to start by analyzing; pairwise alignments then, once the method is understood, it is expanded for multiple sequences. There are two types of dynamic programs, known as global and local alignments. Global alignments assume that the two sequences are similar throughout the protein and align the sequences from end to end. However, since there are bound to be sections of variability, especially in larger sequences, this is not always an adequate assumption.

Local alignments ignore this assumption and look for highly similar regions thoughtout the protein, ignoring the variable regions. This type of alignment works well for specific parts of the protein but does not address overarching similarities in the entire protein.[10]

One global algorithm for determining pairwise functions is known as the Needleman & Wunsch Algorithm. This algorithm traces through the two sequences and minimizes the cost of point mutations and insertions/deletions; it is discussed more thoroughly in Chapter 3. The important part of this algorithm is the cost function:

$$H_{i,j} = \max \left\{ \begin{array}{c} H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - g \\ H_{i,j-1} - g \end{array} \right\}$$

where $S_{i,j}$ is the similarity cost of the two amino acids (point mutation) and g is the cost of a gap (insertion/deletion). The results of the cost function are stored in the cost matrix, H. The top function accounts for point mutations while the other two account for insertions/deletions. To get the alignment, one should backtrack through the matrix starting at the last entry and ending at $H_{2,2}$.[3]

The local complement to the Needleman & Wunsch Algorithm is the Smith-Waterman Algorithm. Although it is very similar to the Needleman & Wunsch, the Smith-Waterman allows for the possibility that the best choice is to stop the alignment. It achieves this by changing the cost function to:

$$H_{i,j} = \max \left\{ \begin{array}{c} H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - g \\ H_{i,j-1} - g \\ 0 \end{array} \right\}.$$

If the maximum value is zero, then that means neither a point mutation nor insertion/deletion is acceptable. The alignment is found by tracing though the algorithm in the same way as Needleman & Wunsch, but the starting point is a maximum and the ending point is a 0.[1]

These dynamic programs can be adapted by changing the matrix from two dimensions to n dimensions. However, this can quickly get overwhelming since the number of possible choices increases exponentially for every additional sequence. A way of overcoming this is using progressive methods. A progressive method is a multiple sequence alignment method that starts with pairwise alignments. After the first pairwise alignment, another sequence is aligned (pairwise) to the current alignment. Additional sequences are added to the alignment. The order that the sequences are

aligned in changes the alignment. This problem can be overcome by grouping sequences by commonalities using binary guide trees and starting with these groups. Additionally, analyzing several pairwise functions and choosing the best to start with helps create a better alignment. Iterative methods start by creating the alignment and then change the order, and the method realigns the sequences again until a local maximum is found. A free-for-non-commercial-use program that uses the progressive method is called ClustalW.[11] This program is similar to the program used in this thesis and therefore will be used to test the program later in the paper.

Another branch of alignments avoids the dynamic programming option and instead uses a heuristics approach. This method is sometimes referred to the "words" method. It makes the assumption that the correct alignment will have sections of highly correlated amino acids. By breaking the sequences into these "words" and comparing them to each other, the correct alignment can be found quickly. However, this method does not deal with specifics, but rather makes general statements about the sequences. The BLAST and FASTA programs use this method and can check a novel sequence to a database. This method is best for making general alignments to find functionality rather than specific alignments that can be used for evolutionary purposes.[9]

Once the sequences are aligned, they can be scored in order to find the optimal alignment. Pairwise alignments are graded one amino acid set at a time. The score is $\sum_{i=1}^{l} s(x_{1i}, x_{2i})$ where l is the length of the alignment and x_{1i} and x_{2i} are the amino acids or gaps in the first and second sequence. If x_{1i} or x_{2i} is a gap, then $s\left(x_{1i}, x_{2i}\right)$ is equal to the gap penalty cost. Otherwise, $s(x_{1i}, x_{2i})$ is equal to a similarity value that is based on the identity of the two amino acids and is looked up in a table. Multiple sequence alignments are scored in a similar manner, but instead $s(x_{1i}, x_{2i})$ is replaced with $\sum_{m=1}^{n} s(x_{mi}, x_{(m+1)i})$ where n is the number of sequences. The final equation is $\sum_{i=1}^{l} \sum_{m=1}^{n} s(x_{mi}, x_{(m+1)i})$.[14][15]

This thesis will explore sequence alignment from the simplest aspect of the dot plot to multiple sequence alignment via the Needleman & Wunsch Algorithm using progressive and iterative methods.

2 The Dot Plot

We first explored the simple dot plot[5]. A dot plot can only evaluate two sequences at a time; however, it provides a quick visual test for similarities. Dot plots can be used to analyze the primary structure of a protein or the nucleotide sequence that codes for a protein. One sequence is placed on the x-axis, and the other one is placed on the y-axis. The first amino acid/nucleic acid in the sequence on the x-axis is compared to the sequence on the y-axis. Each time the same amino acid/nucleic acid appears in the second sequence, a dot is placed on the graph. An added option to a dot plot requires more than one amino acid to be the same before placing a dot. In this program this option is called degrees. For higher degrees, more amino acids/nucleic acids must match up for a dot to be placed. For example, with degrees of 3, three amino acids/nucleic acids in a row must match in order to place a dot at the first of the three. Other models expand this variable of degree into window and threshold. Window refers to the number of pairs checked and threshold is the number that must match. Although these features can quickly be implemented into this program, it was time to move on to more complex models.

Let's look at a small example of how to create a dot plot. Let the first sequence be "THIS" on the x-axis and the second sequence be "THESIS" on the y-axis. Instead of graphing this we are going to use a matrix (D) which looks like this:

$D =$

S				
I				
S				
E				
H				
T				
	T	H	I	S

where the entries are $D_{x,y}$ starting with the bottom left corner as $(0, 0)$. Now we place a dot at $D_{x,y}$ when $D_{x,0} = D_{0,y}$. Since $D_{1,0} = D_{0,1}$ we place a dot at $D_{1,1}$. After all of the dots have been placed, the resulting matrix is as follows:

$D =$

S				•
I			•	
S				•
E				
H		•		
T	•			
	T	H	I	S

Most of the time, looking at the diagonal can show how two sequences are correlated. The more "connected" the diagonal is the more similar the two sequences are. Since this example is not close to a square matrix the idea of the diagonal is not as useful. In this case we are looking for a line from the bottom left corner to the top right corner. The fact that the "diagonal" is disjointed shows us that there was an insertion or deletion; in this case, it was the "ES" in thesis. Now let's use the idea of degrees. When degrees equals two, we need to have two matching sets in a row in order to place a dot. In other words, a dot is placed at $D_{x,y}$ when $(D_{x,0} = D_{0,y}) \wedge ((D_{x+1,0} = D_{0,y+1}) \vee (D_{x-1,0} = D_{0,y-1}))$. Now the modified graph looks like this:

$D =$

S				•
I			•	
S				
E				
H		•		
T	•			
	T	H	I	S

This method promotes dots along the diagonal while reducing the noise. Notice that the cell at $D_{4,4}$, which was previously filled is now blank. This helps us clearly see the disjointed "diagonal". In standard dot plots, degrees is known as the window, and there is a second variable that allows for adjusting the precision known as the mismatch limit. If the window were five and the mismatch limit were two, then as long as there were three matches, a dot would be placed. Because this model does not include the mismatch limit, it was named a modified dot plot.

2.1 An Example Dot Plot

Hemoglobin is a protein in blood that transports oxygen through the body via the bloodstream. It is a tetramer made up of two sets of subunits known as alpha-globulin and beta-globulin. Each subunit has a heme group which coordinates with iron, which has a high affinity for oxygen. A mutation in the beta subunit causes sickle cell anemia. This is caused by the glutamic acid at the 6th position being replaced with a valine.[7] Because of this, hemoglobin subunit beta is both interesting and well documented, which mades it a good protein to analyze.

For the first set of dot plots (see Figure 1), we used primary sequences for subunit beta from a human and a chicken, both found at uniprot.org. The first plot shows the relations between the

sequences without any restrictions, and the second is with the degrees set to 2. From these graphs it is easy to see that there is a correlation between the two sequences based on the main diagonal. The reason there are no disjoint sections of the diagonal is that both sequences are roughly the same length, so most of the variation comes from point mutations. Also, the sequences are long, which makes a single insert hard to notice.

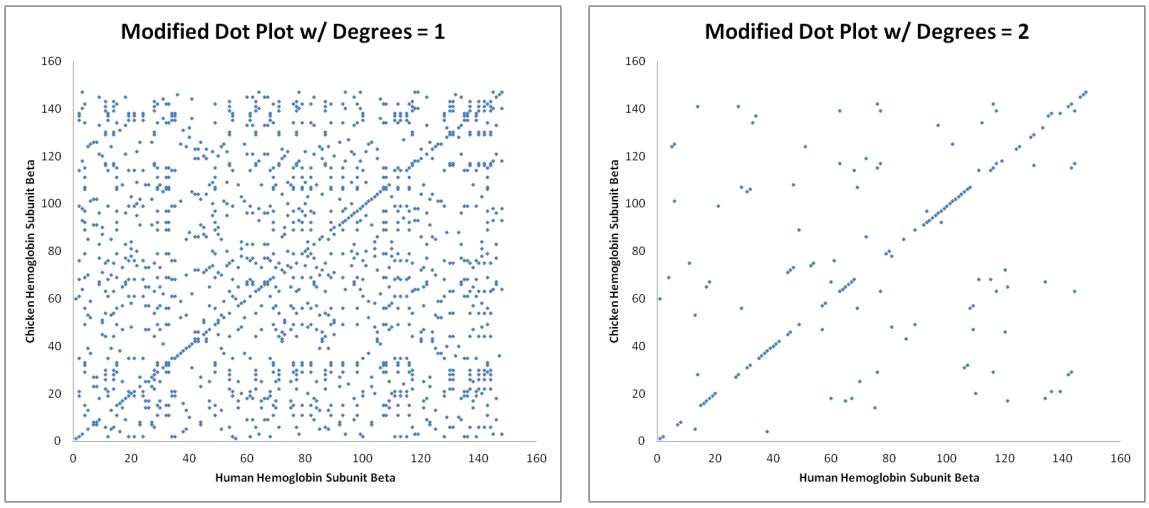

Figure 1: These two dot plots compare human and chicken hemoglobin subunit beta primary sequences with no restrictions (left) and with degrees = 2 (right).

This method quickly shows whether or not two sequences are similar, but does not show enough information to make a conjecture about the function of the proteins or the highly conserved regions. In order to get that information, we would need to compare several different hemoglobin subunit beta sequences to each other and then compare the resulting dot plots. There is also a bad noise problem. This can be handled by the degrees or window and threshold variables, but at the same time this changes the diagonal. It is also possible that those variables can destroy small sections of the diagonal that are shifted. Finally, since the interpretation of the graphs is purely visual, this method is imprecise.

3 NEEDLEMAN & WUNSCH ALGORITHM

The Needleman & Wunsch Algorithm [3] is a bit more sophisticated because it will actually show the optimal alignment based on a cost function that takes into account gaps and substitutions. However, it still can only analyze two sequences at a time. A gap occurs when an amino acid has been added or omitted, and a substitution happens when one amino acid replaces the original. Gaps have a constant cost, and substitutions have a cost based on how similar the two amino acids are, with greater differences having a higher cost.

Using the gap costs and the substitutions costs, the method generates a table using the cost function and then backtracks through the table to find the optimal sequences. It is possible that a branch can occur where two paths seem equally optimal. In this case the method follows both paths to completion and then sums the total cost to find which one cost less if any. We implemented this algorithm in MatLab 7.10.0.

The first step of the method is to create a $N+1$ by $M+1$ matrix called H where N and M are the lengths of the two sequences. This matrix will hold the cost data for every pair of amino acids. The first row and column are reserved for the gap penalties. These entries are found by the equations $H_{1,j} = H_{1,j-1} - g$ for the first row and $H_{i,1} = H_{i-1,1} - g$ for the first column, where g is the gap cost. In order for this to work, we set $H_{1,1} = 0$. All of the other entries are found using the cost function

$$H_{i,j} = \max \left\{ \begin{array}{c} H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - g \\ H_{i,j-1} - g \end{array} \right\}$$

where $S_{i-1,j-1}$ is the similarity cost for the $i-1$ amino acid from the first sequence compared with the $j-1$ amino acid from the second sequence. The $S_{i-1,j-1}$ is found from the BLOSUM40 similarity scores matrices.[4]

The two major types of similarity score matrices are PAM and BLOSUM. PAM stands for Point Accepted Mutation. PAM matrices are based on preexisting global alignments of highly related proteins. The alignments were analyzed for how often one amino acid was replaced with another. The number attached to the name of the matrix refers to the evolutionary distance between two sequences.[13] For example a PAM40 is most sensitive for sequences that are 40 point accepted mutations apart or closely related, while a PAM250 is use for distantly related sequences.[8]

BLOSUM stands for Blocks Substitution Matrix and, like PAM matrices, is based on empirical information. However, to calculate a BLOSUM, local alignments are used. Instead of analyzing the alignment one amino acid at a time as with PAM, BLOSUMs are calculated from chunks of the sequences. These chunks are known as blocks. The number in the name refers to the "minimum percent identity of the blocks used to construct the matrix," or the percentage of the blocks that had to match the standard sequences when calculating the matrix.[13] A higher number means that the sequences are more closely related. BLOSUMs are better for local alignments.

So this information about similarity matrices brings up an interesting question. Why does this model use a BLOSUM40 while at the same time use an algorithm that does global alignments? The answer is simple: this model is to simply get an understanding of the dynamic programming aspect of the model. It will be easy to change which matrix is used later on, and a future program could have an option for which matrix to use. Finally the BLOSUM and PAM matrices are actually closely related. In fact, a BLOSUM45 is nearly equivalent to a PAM250.[8]

Once the matrix is generated, the method starts at the last entry in the matrix and finds where that entry came from. If that entry came from the $H_{i-1,j-1} + S_{i,j}$ part of the cost function, the method moves diagonally and aligns the ith amino acid from the first string with the jth amino acid from the second string. If the entry comes from $H_{i-1,j} - g$, then the string moves up a row and aligns a gap in the first sequence with the jth amino acid of the second sequence. Finally, if the entry came from $H_{i,j-1} - g$, then the method moves left a column and aligns the ith amino acid in the first sequence with a gap in the second sequence. After the method moves through the entire table and ends on the $H_{2,2}$ element, the sequences are fully aligned, and the total cost is the sum of the individual costs of the paths.

Let's do an example using an excerpt from hemoglobin as the first sequence and myoglobin as the second sequence. Myoglobin is similar to hemoglobin except it binds O_2 with higher affinity. This could be because hemoglobin is for transporting oxygen from the lungs to everywhere else, and myoglobin receives the O_2 from hemoglobin.

Sequence 1 = KTEAEMKASEDLKKHGT
Sequence 2 = HGSAQVKGHG

The first step in the algorithm is making the cost matrix. Since the length of sequence 1 is 17 and the length of sequence 2 is 10, the cost matrix will be 18×11 with entries $H_{i,j}$ where the upper left corner is $H_{1,1} = 0$. The first row and column are filled with the gap penalties using the equation $H_{i,1} = -i \cdot g$ and $H_{1,j} = -j \cdot g$ starting at $j = i = 2$ with the gap cost, g, equal to 8. The resulting matrix is shown below.

$$H = \begin{bmatrix} 0 & -8 & -16 & -24 & -32 & -40 & -48 & -56 & -64 & -72 & -80 \\ -8 & & & & & & & & & & \\ -16 & & & & & & & & & & \\ -24 & & & & & & & & & & \\ -32 & & & & & & & & & & \\ -40 & & & & & & & & & & \\ -48 & & & & & & & & & & \\ -56 & & & & & & & & & & \\ -64 & & & & & & & & & & \\ -72 & & & & & & & & & & \\ -80 & & & & & & & & & & \\ -88 & & & & & & & & & & \\ -96 & & & & & & & & & & \\ -104 & & & & & & & & & & \\ -112 & & & & & & & & & & \\ -120 & & & & & & & & & & \\ -128 & & & & & & & & & & \\ -136 & & & & & & & & & & \end{bmatrix}$$

The next step is to start filling in the values using the cost function starting with $H_{2,2}$.

$$H_{2,2} = \max \left\{ \begin{array}{c} H_{1,1} \ + S_{1,1} \\ H_{1,2} - 8 \\ H_{2,1} - 8 \end{array} \right\}$$

First we must look up $S_{1,1}$, which is the similarity value for the first amino acid in sequence 1 (K=lysine) with the first amino acid in sequence 2 (H=histidine). Looking up histidine with lysine in the BLOSUM40 matrix in Appendix A yields a value of -1. Therefore,

$$H_{2,2} = \max \left\{ \begin{array}{c} H_{1,1} \ + S_{1,1} = 0 - 1 = -1 \\ H_{1,2} - 8 = -8 - 8 = -16 \\ H_{2,1} - 8 = -8 - 8 = -16 \end{array} \right\} = -1$$

This process continues to the right and down until the matrix is filled. The final matrix is as follows:

$$H = \begin{bmatrix}
0 & -8 & -16 & -24 & -32 & -40 & -48 & -56 & -64 & -72 & -80 \\
-8 & -1 & -9 & -16 & -24 & -31 & -39 & -42 & -50 & -58 & -66 \\
-16 & -9 & -3 & -7 & -15 & -23 & -30 & -38 & -44 & -52 & -60 \\
-24 & -16 & -11 & -3 & -8 & -13 & -21 & -29 & -37 & -44 & -52 \\
-32 & -24 & -15 & -10 & 2 & -6 & -13 & -21 & -28 & -36 & -43 \\
-40 & -32 & -23 & -15 & -6 & 4 & -4 & -12 & -20 & -28 & -36 \\
-48 & -39 & -31 & -23 & -14 & -4 & 5 & -3 & -11 & -19 & -27 \\
-56 & -47 & -39 & -31 & -22 & -12 & -3 & 11 & 3 & -5 & -13 \\
-64 & -55 & -46 & -38 & -26 & -20 & -11 & 3 & 12 & 4 & -4 \\
-72 & -63 & -54 & -41 & -34 & -25 & -19 & -5 & 4 & 11 & 4 \\
-80 & -71 & -62 & -49 & -42 & -32 & -27 & -13 & -4 & 4 & 8 \\
-88 & -79 & -70 & -57 & -50 & -40 & -35 & -21 & -12 & -4 & 2 \\
-96 & -87 & -78 & -65 & -58 & -48 & -38 & -29 & -20 & -12 & -6 \\
-104 & -95 & -86 & -73 & -66 & -56 & -46 & -32 & -28 & -20 & -14 \\
-112 & -103 & -94 & -81 & -74 & -64 & -54 & -40 & -34 & -28 & -22 \\
-120 & -99 & -102 & -89 & -82 & -72 & -62 & -48 & -42 & -21 & -29 \\
-128 & -107 & -91 & -97 & -88 & -80 & -70 & -56 & -40 & -29 & -13 \\
-136 & -115 & -99 & -89 & -96 & -88 & -78 & -64 & -48 & -37 & -21
\end{bmatrix}$$

The last step in the process is to backtrack through the matrix starting at $H_{18,11}$ and work our way to $H_{2,2}$. We do this by finding where the value for $H_{18,11}$ came from by examining the cost function again.

$$H_{18,11} = \max \left\{ \begin{array}{c} H_{17,10} \ + S_{17,10} = -29 - 2 = -31 \\ H_{18,10} - 8 = -37 - 8 = -45 \\ H_{17,11} - 8 = -13 - 8 = -21 \end{array} \right\}$$

Since $H_{17,11} - 8 = -21$, we know that the value for $H_{18,11}$ came from $H_{17,11}$, and we move to that cell and find where it came from. We continue this process until we reach either $H_{2,2}$, the second row, or the second column. If the second row or column is reached, we just follow it straight left or up respectively to $H_{2,2}$. It is possible for a cell to come from more than one location, resulting in a branch point in which case both paths are examined. The path through the matrix is found below with the branch point in red and the divergent path in green.

$$H = \begin{bmatrix}
0 & -8 & -16 & -24 & -32 & -40 & -48 & -56 & -64 & -72 & -80 \\
-8 & \uparrow & -9 & -16 & -24 & -31 & -39 & -42 & -50 & -58 & -66 \\
-16 & \uparrow & -3 & -7 & -15 & -23 & -30 & -38 & -44 & -52 & -60 \\
-24 & \nwarrow & -11 & -3 & -8 & -13 & -21 & -29 & -37 & -44 & -52 \\
-32 & -24 & \nwarrow & -10 & 2 & -6 & -13 & -21 & -28 & -36 & -43 \\
-40 & -32 & \uparrow & \nwarrow & -6 & 4 & -4 & -12 & -20 & -28 & -36 \\
-48 & -39 & \uparrow & \uparrow & -14 & -4 & 5 & -3 & -11 & -19 & -27 \\
-56 & -47 & -39 & \nwarrow\uparrow & -22 & -12 & -3 & 11 & 3 & -5 & -13 \\
-64 & -55 & -46 & -38 & \nwarrow & -20 & -11 & 3 & 12 & 4 & -4 \\
-72 & -63 & -54 & -41 & \uparrow & -25 & -19 & -5 & 4 & 11 & 4 \\
-80 & -71 & -62 & -49 & -42 & \nwarrow & -27 & -13 & -4 & 4 & 8 \\
-88 & -79 & -70 & -57 & -50 & \uparrow & -35 & -21 & -12 & -4 & 2 \\
-96 & -87 & -78 & -65 & -58 & -48 & \nwarrow & -29 & -20 & -12 & -6 \\
-104 & -95 & -86 & -73 & -66 & -56 & -46 & \nwarrow & -28 & -20 & -14 \\
-112 & -103 & -94 & -81 & -74 & -64 & -54 & -40 & \nwarrow & -28 & -22 \\
-120 & -99 & -102 & -89 & -82 & -72 & -62 & -48 & -42 & \nwarrow & -29 \\
-128 & -107 & -91 & -97 & -88 & -80 & -70 & -56 & -40 & -29 & \nwarrow \\
-136 & -115 & -99 & -89 & -96 & -88 & -78 & -64 & -48 & -37 & \uparrow
\end{bmatrix}$$

If we are in cell $H_{i,j}$, and it came from $H_{i-1,j}$ (a move up), then we pair the $i-1$ amino acid in sequence 1 with a gap. If it came from $H_{i,j-1}$ (a move to the left), then we pair the $j-1$ amino acid in sequence 2 with a gap. If it came from $H_{i-1,j-1}$ (a diagonal move), then we pair the $i-1$ amino acid in sequence 1 with the $j-1$ amino acid in sequence 2.

The resulting sequence alignments for this example as generated from the program are below:

Sequence alignment #1 (blue) is
`KTEAEMKASEDLKKHGT`
`--HGS--A-Q-VKGHG-`
with a score of -21.

Sequence alignment #2 (green) is
`KTEAEMKASEDLKKHGT`
`--HG--SA-Q-VKGHG-`
with a score of -21.

3.1 The Gap Cost

One interesting aspect of this algorithm is the idea of varying gap cost. The program used to get the results above had a gap cost of eight. This gap cost was used because, at the time, it did not seem to matter what cost was assigned to gaps as long as there was a cost, and 8 was the value used in the example that this program was based upon.[3] Once the program was complete and working, it became necessary to justify the value of eight. The first step was to simply change the gap cost and observe the results in the program. The results were not very interesting.

The same primary sequences used in the example were run with various gap costs starting with values higher than eight. No matter how much the gap cost was increased from eight, the resulting sequences were exactly the same except the score dropped, which is to be expected since increasing the gap cost negatively affects the score. At first this led to the hypothesis that gap cost did not matter. This was rationalized by the fact that this program performs a global alignment. This means if there are two sequences of different sizes, enough gaps will be placed in the smaller sequence so that it spans from one side of the larger sequence to the other. Since the sizes of the two sequences does not change based on gap cost, the number of gaps inserted into the smaller sequence will always be the same. In the example above, the smaller sequence will always have seven gaps and they will always be aligned in such a way as to minimize the cost. All gaps cost the same, so what really happens is the amino acids in the smaller sequence are aligned with amino acids in the larger sequence that are the most similar. Changing gap cost will not affect similarity between two amino acids, so the gaps will always be in the same place.

Of course, only costs higher than eight were initially explored, and to ignore values smaller would be poor practice. The negative numbers were excluded simply because that would be analogous to rewarding gaps which biologically makes no sense because gaps are less likely than swaps. However, when $1 < g < 8$, the same alignments resulted. These results were far more interesting. When the gap cost was one, seven different "optimal" alignments were created, two of which were the originals from the example. These are only optimal based on our definition of the scoring system even though they are not optimal on a biological sense. The other five had a gap placed in the longer sequence. For example:

$g = 1$	$g = 0$
Sequence alignment #5 is KTEAEMKASEDLK-KHGT --HGS--A-Q-VKG-HG- with a score of 28.	Sequence alignment #31 is K--TEAEMKASEDLK-KHGT -HGS-AQ------VKG-HG- with a score of 38.

When the gap cost was zero, 33 different "optimal" alignments were created, with either 2 or 3 gaps placed in the larger sequence. Interestingly, the original alignments were not part of the 33; neither were any from when gap cost was one. An example from this test is displayed above. In both of the examples the score is higher than the score for the original alignments which was -21. This is a problem for several reasons: lower gap costs lead to a larger number of "optimal" alignments, "gap-ier" alignments, and alignments with supposedly higher scores.

The first problem of more alignments is really only a computing problem. More alignments means more iterations, which leads to longer processing time. If a gap cost that reduces the number of alignments can be justified, then that will reduce problems down the road for multiple sequence alignments. This high gap cost is justified by the next two problems. The problem with "gap-ier" sequences is that nature does not like gaps. This is because gaps have a higher chance of creating a malfunction protein which leads to the death of the organism. It is not that gaps are unlikely; it is that gaps are deadly, and therefore the occurrence of gaps in living populations is low. Basically, we need to use a gap cost that is high enough to exclude gaps when a swap is possible, while at the same time allow for a gap if it helps reduce the overall score.

Looking for an optimal score, leads to problem three: the apparent higher scores with lower gap costs. Alignments using different gap costs cannot be compared with each other using their respective scores because each score was calculated with a different gap cost. Having a lower gap cost will increase the score for all alignments equally. The only way to compare two alignments with different gap costs is to recalculate the alignment's score using an universal gap cost. If both scores from the alignments with $g = 1$ or 0 are recalculated with $g = 8$ the resulting scores are -35 and -62 respectively, meaning that the original sequences are probably the more optimal sequences. Another way of looking at it is from the fact that increasing the gap cost from eight does not change the alignments. In a way, they have reached equilibrium. Having the gap cost of eight seems to be the answer for this situation.

On a small scale, the problem of gap cost is somewhat resolved. However, this problem can be further explored to pick a gap cost that works on a more universal scale. To do this, we have to look at how the program deals with gaps, and that falls on the BLOSUM40 table. When $g = 1$, the example alignment places a gap in the 14^{th} position aligned with a G. Normally (when $g = 8$) that G is aligned with a K. The BLOSUM40 similarity score for a swap from a G to a K is -2. When the alignment is created, if a swap is chosen, the total score will be decreased by 2. However, if a gap is aligned with the G and a second gap is aligned with the K, as with sequence alignment #5, the total score will be decreased by $2g$. When the gap cost is one, the score is decreased by two which is equally optimal to performing a swap from G to K. This causes the program to find both alignments optimal. So in the case of the myoglobin and hemoglobin sequence, setting the gap cost higher than one will prevent unnecessary "gap-iness."

If we use this same logic, we can come up with a value for the gap cost that will prevent "gap-iness" for most sequences. Looking at the BLOSUM40 table the highest penalty is a switch from a W to a C with a cost of -6. This means when the gap cost is three, it is equally optimal to swap a W for a C as it is to place two gaps, giving three possibilities.

Swap C with W	Gap starts in the first sequence	Gap starts in the second sequence
...C... ...W...	...−C... ...W−...	...C−... ...−W...

Considering that placing two gaps is biologically irrelevant since it artificially increases the number of gaps, we want to choose a gap cost greater than three to prevent this situation. At the same time, we want to place the two gaps if the resulting alignment is more stable. In the BLOSUM40 table, the best score occurs when a W is aligned with a W for a score of 19. If we were trying to align two sequences, ...CW... and ...WC..., the best alignment would be to align the two Ws together and align the Cs with a gap, assuming that this snippet happens at about the same location in each sequence. The resulting alignment should be
...CW−...
...−WC...

However, if we make the gap cost too high, then this situation will not be favored. With both of these possibilities in mind, we want $3 < g < 19$. A safe bet would be to take the average number in that range rounded to the nearest integer, which is 10. Interestingly enough, the default value for the gap cost when using ClustalW is also 10.[2] As stated before, ClustalW is a similar alignment program and is therefore a good benchmark for comparison. In reality, the value of the gap cost is up to the user as well as which comparison matrix is used. Although 10 is probably a useful value, it is just a guideline and all alignments using any gap cost must be appropriately interpreted.

3.2 Gap Extensions

As has been said many times before, nature does not like gaps. Because of this, it is more likely that a string of gaps will occur together rather than the same number of gaps interspersed throughout the sequence. For example, imagine an alignment produces one sequence that has a total of five gaps. There are several equally optimal alignments, one with the five gaps in a row, another with two gaps followed by a string amino acids and the other three gaps and more. The one with five

gaps in a row represents a massive insertion or deletion event, while the other alignment represents two smaller insertions or deletion events. Because insertions and deletions are rare, it is more likely that a huge chunk was removed or added than several small chunks. This is why the situation with five gaps in a row should be favored. This is achieved through the gap extension penalty.

A gap extension happens when a new gap is aligned next to another gap extending the gapped area. To favor this situation, the cost for each additional gap is reduced. There are several ways of achieving this. One is to reduce the gap cost by some constant each time another gap is added, thus creating a compound benefit to an extended gap. Another way is to have a separate gap cost for extended gaps. This is the method chosen for this program. Like gap cost, the gap extension cost can also be set by the user. The alignments are calculated normally, but when the alignments are scored, the gap extension is factored in to the calculation. Every time a gap is detected in either sequence, the program checks the previous amino acid to see if it is also a gap. If it is, then only the gap extension cost is subtracted from the total score. If the previous amino acid is in fact an amino acid, then the original gap cost is used.

Using the gap extension cost can reduce the number of alignments, which will make computing multiple sequence alignments faster. An example of gap extension occurs with the sequences `FTFTALILLAVAV` and `FTFALIAV`. When they are aligned without a gap extension factored in, we get

Sequence alignment #1 is	Sequence alignment #2	Sequence alignment #3 is
`FTFTALILLAVAV`	`FTFTALILLAVAV`	`FTFTALILLAVAV`
`FTF-ALI--AV--`	`FTF-ALI--A--V`	`FTF-ALI----AV`
with a score of 1.	with a score of 1.	with a score of 1.

Notice how all of the sequences have the same score of one. Although it is possible for any one of these sequences to be "correct," alignment #3 is more likely to be biologically significant. When these sequences are reanalyzed using the gap extension cost of one, we get scores of 19, 19, and 28 respectively, showing that #3 is favored.

Now the question becomes the following: what value should be used for the extension cost? If the value is lower than the gap cost, extensions will be favored, and if it is above the gap cost, then extensions will be penalized. Therefore, we want the cost to be below the gap cost. As for what

value below the gap cost we want to use, it does not really matter. Any value below the gap cost will create a difference in score between extended gaps and interspersed gaps. The only difference between using an extension value of one or nine is that one will create a more drastic difference and nine will create a more subtle difference. When it comes to programming, however, any difference will be detected regardless of magnitude. For this program, the extension cost is set to one to get the more drastic changes.

3.3 Real/Long Sequences

Now that the two gap costs are set, it is time to put the program to the test with real sequences, not just snippets. The protein chosen for this is cytoplasmic polyadenylation element binding protein 1 or CPEB1. This protein was chosen because it is more variable in length among species than hemoglobin, and it is a long protein (>560 amino acids). CPEB1 is a protein that induces the addition of the poly-A tail to mRNA. In a sense, CPEB1 promotes translation.[**?**][12] Using the UniProt database for proteins, we obtained the primary sequences of CPEB1 for a human and a mouse and then used our program to align them. The resulting alignment is below. The alignment is split into nine sections for easy reading with the top sequence in each sequence from the human version of the protein and the bottom sequence from the mouse.

Sequence alignment #1 is as follows:

```
MALSLEEEAGRIKDCWDNQEAPALSTCSNANIFRRINAILDNSLDFSRVCTTPINRGIHDHLPDFQDSEE
MAFSLEEESGRIKDCWDNQEVPALSTCSNANIFRRINAILDDSLDFSKVCTTPINRGIHDQLPDFQDSEE

TVTSRMLFPTSAQESSRGLPDANDLCLGLQSLSLTGWDRPWSTQDSDSSAQSSTHSVLSMLHNPLGNVLG
AITSRMLFPTSAQESPRGLPDANGLCLGLQSLSLTGWDRPWSTQDSDSSAQSNTQSVLSMLQNPLGNVLG

KPPLSFLPLDPLGSDLVDKFPAPSVRGSRLDTRPILDSRSSSPSDSDTSGFSSGSDHLSDLISSLRISPP
KTPLSFLSLDPLGSDL-DKFPAPSVRGSRLDTRPILDSRSSSPSDSDTSGFSSGSDHLSDLISSLRISPP

LPFLSLSGGGPRDPLKMGVGSRMDQEQAALAAVTPSPTSASKRWPGASVWPSWDLLEAPKDPFSIEREAR
LPFLSMTGNGPRDPLKMGVGSRMDQEQAALAAVAPSPTSAPKRWPGTSVWPSWDLLGAPKDPFSIEREAR

LHRQAAAVNEATCTWSGQLPPRNYKNPIYSCKVFLGGVPWDITEAGLVNTFRVFGSLSVEWPGKDGKHPR
LHRQAAAVNEATCTWSGQLPPRNYKNPIYSCKVFLGGVPWDITEAGLVNTFRVFGSLSVEWPGKDGKHPR

CPPKGNMPKGYVYLVFELEKSVRSLLQACSHDPLSP-GLSEYYFKMSSRRMRCKEVQVIPWVLADSNFVR
CPPKGNMPKGYVYLVFELEKSVRALLQACSHDPLSPDGLSEYYFKMSSRRMRCKEVQVIPWVLADSNFVW
```

```
SPSQRLDPSRTVFVGALHGMLNAEALAAILNDLFGGVVYAGIDTDKHKYPIGSGRVTFNNQRSYLKAVSA
SPSQRLDPSRTVFVGALHGMLNAEALAAILNDLFGGVVYAGIDTDKHKYPIGSGRVTFNNQRSYLKAVTA

AFVEIKTTKFTKKVQIDPYLEDSLCHICSSQPGPFFCRDQVCFKYFCRSCWHWRHSMEGLRHHSPLMRNQ
AFVEIKTTKFTKKVQIDPYLEDSLCLICSSQPGPFFCRDQVCFKYFCRSCWHWRHSMEGLRHHSPLMRNQ

KN----
KNRDSS
```

with a score of 4119.

This large alignment helps better visualize the need for gaps. The sections in red show where one sequence is aligned with a gap in the other sequence. If the gaps were not placed appropriately, then it is easy to see how the two sequences would be very misaligned. For example if the first gap that is aligned with a V in the human protein was not inserted, then the following sequence of DKFPAPS... would be shifted down by one amino acid. This means even though the two sequences are exactly the same for 58 amino acids after the gap, they would not be aligned as such.

Another feature to notice is the swaps that are labeled blue. If any of these swaps were aligned with a gap rather than the swap, then that would shift the sequence over, ruining the entire alignment. From this, we can say that our gap cost value is doing a fairly adequate job. At the same time the gap cost has to be far above 10 to cause problems with the alignment. Above around $g = 500$, the alignment changes by dispersing the four gaps at the end throughout the alignment. This is clearly not the best alignment because moving gaps in one sequence will shift large sections of amino acids away from their identical counterparts in the other sequence. As for lowering the gap cost, we get the same alignment as with the gap cost at 10 until $g = 1$ or 0. At that point, the number of possible sequences explodes and the program enters a near infinite loop and never finishes.

The last experiment performed with these sequences was to play the role of nature by deleting large sections from the mouse's protein. The point of this was to see if the program would accurately place gaps where the sections were deleted. When the modified sequences were used in the program, the results showed exactly which sections were removed. The two sections were "PIYS" and "PPKG" from the mouse's sequence, which are displayed in green in the previous alignment. The

sequences were run through the program again, and the alignment was the same except for those two sections. This is a 70-amino-acid excerpt from that alignment that shows the two sections. Again the top sequence is from the human protein and the bottom is the mouse.

```
PPRNYKNPIYSCKVFLGGVPWDITEAGLVNTFRVFGSLSVEWPGKDGKHPRCPPKGNMPKGYVYLVFELE
PPRNYKN----CKVFLGGVPWDITEAGLVNTFRVFGSLSVEWPGKDGKHPRC----NMPKGYVYLVFELE
```

Finally, the program's alignments were tested alongside ClustalW. The original two proteins were also aligned using the ClustalW program, and the resulting alignment is identical to the original alignment. ClustalW was also used to analyze the modified proteins, and again the two alignments were identical. With confidence that the program is running smoothly and that the parameters are correct, it is time to make a multiple sequence alignment program.

4 MULTIPLE SEQUENCE ALIGNMENT PROGRAM

The idea behind the multiple sequence alignment program was rather simple. There would be a main program that would handle which sequences to feed into a subroutine that would use the mechanics from method 2 to create a pairwise alignment. Each pairwise alignment would be fitted to the previous alignments. The original idea was to have the user input the sequences in order of similarity or have the program determine this order. Then the main program would take the first two sequences, use method 2 to align them and save the optimal alignment, and take the next sequence and align it to the second sequence in the previous alignment.

For example, say we want to align sequences A, B, C, and D. The program determines that B and D are the most similar followed by C then A. The order that the program would feed the sequences into the subroutine would be B, D, C, A. The main program would take sequences B and D and run them through method 2 resulting in B′ and D′ which are the aligned versions of B and D including gaps. If D was originally THSE and after it was aligned, it had a gap between H and S, then D′ would be TH-SE. After B and D are aligned, C is aligned with D′ to get C′. Finally, A would be aligned with C′ and the multiple sequence alignment would be B′, D′, C′, and A′. The multi-score is calculated in the way described in the introduction.

There were several problems with this idea. First of all, each pairwise alignment could produce numerous optimal alignments, so each would have to be explored in order to find the optimal multiple alignment. If the alignment of B and D produced 4 optimal alignments, then C would have to be aligned with all four of the D′ alignment sequences. Each one of these could produce more than one optimal sequence and A would have to be aligned with all of them. Quickly, this method can get out of hand.

The other problem occurs when an aligned sequence is larger than the previous aligned sequence. In reference to the example, the problem occurs when C′ is larger than B′. This is a problem when the sequences get placed in the multiple sequence alignment of B′, D′, C′, and A′. If C′ is 50 characters long and B′ is 48 characters long, then there will be a disconnect between B′ and D′ for two reasons. First, which alignment of D′ is correct? Is it the one aligned with B or

the one aligned with C? They are both different because at the very least they are different lengths. Second, if the alignment with C is chosen, then every sequence after that will be aligned according to the new length of D′ and the original alignment of B becomes irrelevant. Let B=THS, D=THSE, C=THESE, and A=THERE. Then B aligned with D will be

$$\begin{array}{lcl} B' & = & \texttt{THS-} \\ D'_1 & = & \texttt{THSE} \end{array}$$

However, D aligned with C will be

$$\begin{array}{lcl} D'_2 & = & \texttt{TH-SE} \\ C' & = & \texttt{THESE} \end{array}$$

If D'_2 is chosen then the final multiple sequence alignment would be

```
THS--
TH-SE
THESE
THERE
```

This problem could be solved by recalculating the previous alignments whenever the length changes for both versions of D′. However, this creates new problems by possibly increasing the number of branch points and thus further complicating the system.

To get around these problems, method 3 starts by sorting the sequences by lengths. Then it aligns the two longest sequences. Each additional sequence is aligned to the longest sequence. The longest sequence is now the reference sequence. If at any point the length of the reference sequence is increased by adding gaps, the lengthened version of the reference sequence becomes the new reference sequence and the program starts over with the new reference. Every time a pairwise sequence is analyzed, all resulting alignments are recorded, and at the end of the multiple sequence alignment, every combination of these pairwise alignments is used to create a number of multiple sequence alignments, and their individual multi-scores are shown to find the best multiple alignment. Using the example from before, the order that the sequences would be aligned is C, A, D, B. C and A would be aligned first to get C′ and A′. If C′ is larger than C, then the program starts over by aligning C′ with A. After C and A are properly aligned, D would be aligned with C to get D′ and so on. The resulting sequence is shown below.

```
THESE
THERE
TH-SE
TH-S-
```

This method works fairly well, but since it only aligns with the longest sequence, it is possible that two individual sequences, other than the first, could be better aligned. This occurs more often when there are a few sequences that are very different from the reference sequence. In that case it would be better to align those sequences together, then as a group align them to the reference sequence.

To test the program, we switched back over to hemoglobin and myoglobin because there are multiple versions of these proteins and unlike CEPB1, the sequence varies greatly from version to version providing more opportunities for branch point and numerous different multiple alignments. The proteins chosen were the Alpha and Beta subunits of hemoglobin and myoglobin from humans. While looking for the myoglobin sequence, a search revealed a protein labeled Uncharacterized Protein from a giant panda. The sequence seemed similar to human myoglobin, so it was added to the test multiple sequence alignment to see how similar. The results of the multiple sequence alignment are below. The program produced eight multiple alignments and only the optimal one is shown. As before the alignment is broken into sections. The order of each section is Alpha, Beta, Myoglobin, Giant Panda.

Multiple Sequence Alignment #3 is

```
M-VLSPADKTNVKAAWGKVGAHAGEYGAEAL--ERMFLSFP-TTKTY--FPHF---DLSHGSAQVKGHGK
MVHLTPEEKSAVTALWGKVNVDEVG-G-EAL-G-RLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGK
M-GLSDGEWQLVLNVWGKVEADIPGHGQEVLI--RLFKGHPETLEKFDKFKHLKSEDEMKASEDLKKHGA
M-GLSDGEWQLVLNVWGKVEADLAGHGQEVLISHRLFKGHPETLEKFDKFKHLKSEDEMKGSEDLKKHGN

KVADALTNAVAHVDDMPNALSALSDLHAHK-LRVDPVNF-KLLSHCLLVTLAAHLPA-EFTPAVHASLDK
KVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVD-PENF-RLLGNVLVCVL-AHHFGKEFTPPVQAAYQK
TVLTALGGILKKKGHHEAEIKPLAQSHATK-HKI-PVKYLEFISECIIQVLQSKHPG-DFGADAQGAMNK
TVFTALGGILKKKGQHEAELKPLAQSHATK-HKI-PVKYLEFISEAIIQVLQSKHPG-DFGADAQAAMRK

FLASVSTVLTSKYR------
VVAGVANALAHKY-----H-
ALELFRKDMASNYKELGFQG
ALELFRNDIAAKYKELGFQG
```

with a score of 1470.

The reference sequence for this alignment was the giant panda protein, so every other sequence was

aligned with that one. A result is that the myoglobin sequence aligned fairly well, but the alpha and beta subunits aligned poorly to themselves. This is probably because hemoglobin subunits and myoglobin are too different to get useful information from a multiple alignment with both types. When just the two subunits are aligned, we get the following results with the alpha on top and the beta on the bottom:

```
MV-LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-----HGSAQVKGHGKKV
MVHLTPEEKSAVTALWGKVNVD--EVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKV

ADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASV
LGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGV

STVLTSKYR
ANALAHKYH
```

An example of the subunits' aligning poorly occurs within the first three amino acids. The alpha subunit is aligned M-VL but the beta subunit is MVHL. A better alignment would be to place the gap on the other side of the V in the alpha subunit. The reason is that V is actually being aligned with the G in the giant panda sequences. The similarity score for V to G is -4 and the cost of aligning V with the gap in the panda sequence is -10. Technically, that means this is the optimal alignment, but we now have the question of whether it is biologically meaningful. This depends on the question that is being asked. If the question is how similar the subunits are to myoglobin, then we would want to get the subunit alignment first and then align myoglobin to the subunit alignment as a whole. In other words, we set up the alignment using groups of similar sequences rather than align them all independently. This would insure that the more similar sequences have common alignments. Aligning them independently would be helpful when a sequence with an unknown purpose is aligned to sequences of known purpose to determine if the unknown sequences are related. From this, it could be hypothesized that the unknown sequence might have the same function as the known sequences it was compared with.

Running the same four proteins through ClustalW gets the following alignment:

```
ALPHA        MV-LSPADKTNVKAAWGKVGAHAGEYGAEALE--RMFLSFPTTKTYFPHF-DLSH-----
BETA         MVHLTPEEKSAVTALWGKV--NVDEVGGEALG--RLLVVYPWTQRFFESFGDLSTPDAVM
MYOGLOBIN    -MGLSDGEWQLVLNVWGKVEADIPGHGQEVLI--RLFKGHPETLEKFDKFKHLKSEDEMK
GIANTPANDA   -MGLSDGEWQLVLNVWGKVEADLAGHGQEVLISHRLFKGHPETLEKFDKFKHLKSEDEMK

ALPHA        GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLA
BETA         GNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLA
MYOGLOBIN    ASEDLKKHGATVLTALGGILKKKGHHEAEIKPLAQSHATKHKIPVKYLEFISECIIQVLQ
GIANTPANDA   GSEDLKKHGNTVFTALGGILKKKGQHEAELKPLAQSHATKHKIPVKYLEFISEAIIQVLQ

ALPHA        AHLPAEFTPAVHASLDKFLASVSTVLTSKYR------
BETA         HHFGKEFTPPVQAAYQKVVAGVANALAHKYH------
MYOGLOBIN    SKHPGDFGADAQGAMNKALELFRKDMASNYKELGFQG
GIANTPANDA   SKHPGDFGADAQAAMRKALELFRNDIAAKYKELGFQG
```

Needless to say, there is not much in common between ClustalW's output and method 3's output. This is somewhat expected since method 3 only takes into account the longest sequence when calculating the alignment. However, the alpha and beta alignments from ClustalW and the alignment from our program are very similar. Some aspects of the ClustalW alignment need examining. Notice that the first amino acid in every sequence is an M, but these are not all aligned with each other. The myoglobin half are aligned with the V in the subunit's half. It seems that the better alignment might be

```
MV-LS
MVHLT
M-GLS
M-GLS
```

The main problem with both alignments might just be that the subunits are too different from myoglobin to get useful results. Either way, it is clear that there could be vast improvements to method 3.

5 FUTURE IMPROVEMENTS/CONCLUSIONS

There are many features that could be added to method 3 that would vastly improve its speed, accuracy, and relevance. A few are smarter gap extension calculation, comparing all sequences in a multiple alignment to determine similarity, and aligning based on the reference sequence as well as the most similar sequence. There could be features that add to the customizability of the program such as multiple comparison tables to choose from and a drop down menu.

Right now the gap extension penalty is factored into the score calculation after the alignments are already found. It should be possible to modify the objective function from the Needleman & Wunsch Algorithm (below) to check if the previous step was a gap. In the objective function, the terms involving g result in a gap in the sequence, so all the program would have to do is check if one of those terms is used and then keep track of it.

$$H_{i,j} = \max \left\{ \begin{array}{c} H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - g \\ H_{i,j-1} - g \end{array} \right\}$$

This could be done by first minimizing the objective function, $H_{i,j}$. If the minimum is one of the gap equations, then, depending on which equation from the pairwise function was used to find the minimum, the program can determine which cell in the matrix came before the current cell. If that previous cell came from a gap as well, then the current cell could subtract the gap extension cost rather than the gap cost.

The next advancement would be to start a multiple alignment by doing several pairwise alignments and keeping track of the scores. Then the scores could be used to group alignments by similarity. The sequences from these groups could be run through the multiple alignment program one group at a time. Finally each group as a whole could be aligned to the other groups, giving a multiple alignment based on similarity rather than length.

Eventually the number of possible sequence alignments will reach a level that will take exponentially more time. This is where making the program smarter would come in handy; in other words: the program should use heuristics. Whenever doing a multiple sequence alignment by aligning the next sequence with the previous sequence (rather than the reference), there will be multiple pairwise alignments, but only one can be used to complete the multiple alignments. The

program could do several iterations starting with first randomly choosing which pairwise alignments to choose. Then, after the alignment is finished, the program could determine whether its random choices were optimal based on the scoring algorithm. The program could then repeat the alignment but this time change one of the original pairwise alignments. If the final score is lower, the program should revert back and choose a different set of initial conditions. If the program allows the user to set the number of iterations allowed, then accuracy is determined by the user. Theoretically, given infinite iterations, the global maximum can be found and with limited iterations, local maxima can be found.

In conclusion, the pairwise alignment from method 2 is rather reliable, while the multiple alignment program needs work. In reality, there is not a huge need for multiple alignments that show the amino acid – amino acid pairs. The majority of the information acquired from alignments is simply knowing which regions are similar and to what degree. To do this, alignments using "words" are far more useful and faster. This is why database programs such as BLAST and FASTA are so popular. When it comes to amino acid – amino acid pairings, the pairwise alignments are more useful since most of the time the information one is looking for is more localized to one section of the protein such as functional domains. This lowers the size of the sequences making pairwise alignments easier to handle.

At a recent presentation, Michael Waterman (from the Smith-Waterman Algorithm) discussed a new way of comparing sequences known as D_2. D_2 is calculated by examining each sequence for the occurrence of every possible two-letter "word" in each sequence.[17] For example, if the sequence is ATAT and the two-letter word is "AT," then that sequence has two occurrences. Every two-letter "word" means every two-letter combination taken from the"alphabet" that was used to create the sequence. The algorithm scans down the sequence keeping track of the two letter words. The final value for D_2 can be used as a statistic to compare sequences. Sequences with similar values for D_2 are likely to have similar primary sequences. It is possible that this statistic could revolutionize how sequence alignments are performed.

REFERENCES

[1] "Alignment Algorithms," Molecularsciences.org; available at <http://www.molecularsciences.org/bioinformatics/pairwise_alignment/alignment_algorithms>

[2] "Align Sequences using ClustalW2," European Bioinformatics Institute; available at <http://www.ebi.ac.uk/Tools/msa/clustalw2/>

[3] Felix Autenrieth et al., "Sequence Alignment Algorithms," University of Illinois at Urbana-Champaign; available at <http://www.ks.uiuc.edu/Training/SumSchool/materials/sources/tutorials/07-bioinformatics/seqlab-html/node6.html>

[4] "BLOSUM40 amino acid substitution matrix," California State University - Los Angeles; available at <http://www.calstatela.edu/faculty/jmomand/BLOSUM40.txt>

[5] R. A. Bowen, "Background on Nucleic Acid Dot Plots," Colorado State University - Molecular Toolkit; available at <http://www.vivo.colostate.edu/molkit/dnadot/bkg.html>

[6] Scott Freeman, *Biological Science,* Benjamin-Cummings Pub Co, 2007.

[7] "HBB," Genetics Home Reference; available at <http://ghr.nlm.nih.gov/gene/HBB>

[8] "Help with Matrices Used in Sequence Comparison Tools," European Bioinformatics Institute; available at <http://www.ebi.ac.uk/help/matrix.html>

[9] "Heuristic Algorithms," Molecularsciences.org; available at <http://www.molecularsciences.org/bioinformatics/pairwise_alignment/heuristic_algorithms>

[10] "Pairwise local/global alignment - Introduction," European Bioinformatics Institute; available at <http://www.ebi.ac.uk/2can/tutorials/protein/align.html>

[11] "Progressive Alignment Methods," Molcularsciences.org; available at <http://www.molecularsciences.org/bioinformatics/multiple_sequence_alignment/progressive_alignment_methods>

[12] J. Richter, CPEB: a life in translation. *Trends Biochem. Sci.* 32 (2007) 279-285.

[13] Keith Robison, "Widely Used Matrices" Harvard Molecular Technology Group & Lipper Center for Computational Genetics; available at <http://arep.med.harvard.edu/seqanal/submatrix.html>

[14] "Scoring Model," Molecularsciences.org; available at <http://www.molecularsciences.org/bioinformatics/pairwise_alignment/scoring>

[15] "Scoring MSA," Molecularsciences.org; available at <http://www.molecularsciences.org/bioinformatics/multiple_sequence_alignment/scoring_msa>

[16] Universal Protein Resource (UniProt); available at <http://www.uniprot.org/>

[17] Michael Waterman, "Sequence Comparison without Alignment," University of Southern California; March 2, 2002.

APPENDIX A: BLOSUM40 TABLE[4]

	A	B	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	X	Y	Z	*
A	5	-1	-2	-1	-1	-3	1	-2	-1	-1	-2	-1	-1	-2	0	-2	1	0	0	-3	0	-2	-1	-6
B	-1	5	-2	6	1	-3	-1	0	-3	0	-3	-3	4	-2	0	-1	0	0	-3	-4	-1	-3	2	-6
C	-2	-2	16	-2	-2	-2	-3	-4	-4	-3	-2	-3	-2	-5	-4	-3	-1	-1	-2	-6	-2	-4	-3	-6
D	-1	6	-2	9	2	-4	-2	0	-4	0	-3	-3	2	-2	-1	-1	0	-1	-3	-5	-1	-3	1	-6
E	-1	1	-2	2	7	-3	-3	0	-4	1	-2	-2	-1	0	2	-1	0	-1	-3	-2	-1	-2	5	-6
F	-3	-3	-2	-4	-3	9	-3	-2	1	-3	2	0	-3	-4	-4	-2	-2	-1	0	1	-1	4	-4	-6
G	1	-1	-3	-2	-3	-3	8	-2	-4	-2	-4	-2	0	-1	-2	-3	0	-2	-4	-2	-1	-3	-2	-6
H	-2	0	-4	0	0	-2	-2	13	-3	-1	-2	1	1	-2	0	0	-1	-2	-4	-5	-1	2	0	-6
I	-1	-3	-4	-4	-4	1	-4	-3	6	-3	2	1	-2	-2	-3	-3	-2	-1	4	-3	-1	0	-4	-6
K	-1	0	-3	0	1	-3	-2	-1	-3	6	-2	-1	0	-1	1	3	0	0	-2	-2	-1	-1	1	-6
L	-2	-3	-2	-3	-2	2	-4	-2	2	-2	6	3	-3	-4	-2	-2	-3	-1	2	-1	-1	0	-2	-6
M	-1	-3	-3	-3	-2	0	-2	1	1	-1	3	7	-2	-2	-1	-1	-2	-1	1	-2	0	1	-2	-6
N	-1	4	-2	2	-1	-3	0	1	-2	0	-3	-2	8	-2	1	0	1	0	-3	-4	-1	-2	0	-6
P	-2	-2	-5	-2	0	-4	-1	-2	-2	-1	-4	-2	-2	11	-2	-3	-1	0	-3	-4	-2	-3	-1	-6
Q	0	0	-4	-1	2	-4	-2	0	-3	1	-2	-1	1	-2	8	2	1	-1	-3	-1	-1	-1	4	-6
R	-2	-1	-3	-1	-1	-2	-3	0	-3	3	-2	-1	0	-3	2	9	-1	-2	-2	-2	-1	-1	0	-6
S	1	0	-1	0	0	-2	0	-1	-2	0	-3	-2	1	-1	1	-1	5	2	-1	-5	0	-2	0	-6
T	0	0	-1	-1	-1	-1	-2	-2	-1	0	-1	-1	0	0	-1	-2	2	6	1	-4	0	-1	-1	-6
V	0	-3	-2	-3	-3	0	-4	-4	4	-2	2	1	-3	-3	-3	-2	-1	1	5	-3	-1	-1	-3	-6
W	-3	-4	-6	-5	-2	1	-2	-5	-3	-2	-1	-2	-4	-4	-1	-2	-5	-4	-3	19	-2	3	-2	-6
X	0	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-2	-1	-1	0	0	-1	-2	-1	-1	-1	-6
Y	-2	-3	-4	-3	-2	4	-3	2	0	-1	0	1	-2	-3	-1	-1	-2	-1	-1	3	-1	9	-2	-6
Z	-1	2	-3	1	5	-4	-2	0	-4	1	-2	-2	0	-1	4	0	0	-1	-3	-2	-1	-2	5	-6
*	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	1

APPENDIX B: NEEDLEMAN & WUNSCH ALGORITHM CODE

A line starting with "///" indicates a continuation of the previous line.

```
%        newvalues = input('Do you want to use new values?','s');
%        if isempty(newvalues)
%             newvalues = 'n';
%        end
%        if newvalues == 'y'
%             String1=input('Sequence #1:','s');
%             String2=input('Sequence #2:','s');
%             gap=input('Gap Penalty:');
%        end
%        display=input('Top n or All: ','s');
% if exist('display','var')==0
%     display='all';
% end
% if exist('String1','var')==0
%     String1='empty';
% end
% if exist('String2','var')==0
%     String2='empty';
% end
% if exist('gap','var')==0
%     gap='8';
% end
%
% if isempty(String1)
%     String1='empty';
% end
% if isempty(String2)
%     String2='empty';
% end
% if isempty(gap)
%     gap='8';
% end
% if isempty(display)
%     display='all';
% end
%        prompt = {'Sequence #1:','Sequence #2:','Gap Penality:','Top n
///or All:'};
%        dlg_title = 'Jeffs Sequence Alignment Program';
%        num_lines = 1;
%        def = {String1,String2,num2str(gap),display};
%        answer = inputdlg(prompt,dlg_title,num_lines,def);
%        if isempty(answer)
%            return
%        end
%
%        String1=char(answer(1));
```

```
%        String2=char(answer(2));
%        gap=str2double(answer(3));
%        display=char(answer(4));

       String1=current;
       String2=char(Seqlist(next));

       gapgap=0;

       %Comparison table
BSTABLE = [5 -2 -1 -1 -2  0 -1  1 -2 -1 -2 -1 -1 -3 -2  1  0 -3 -2  0 -1
///-1  0 -gap;
          -2  9  0 -1 -3  2 -1 -3  0 -3 -2  3 -1 -2 -3 -1 -2 -2 -1 -2 -1
///0 -1 -gap;
          -1  0  8  2 -2  1 -1  0  1 -2 -3  0 -2 -3 -2  1  0 -4 -2 -3  4
///0 -1 -gap;
          -1 -1  2  9 -2 -1  2 -2  0 -4 -3  0 -3 -4 -2  0 -1 -5 -3 -3  6
///1 -1 -gap;
          -2 -3 -2 -2 16 -4 -2 -3 -4 -4 -2 -3 -3 -2 -5 -1 -1 -6 -4 -2 -2
///-3 -2 -gap;
           0  2  1 -1 -4  8  2 -2  0 -3 -2  1 -1 -4 -2  1 -1 -1 -1 -3  0
///4 -1 -gap;
          -1 -1 -1  2 -2  2  7 -3  0 -4 -2  1 -2 -3  0  0 -1 -2 -2 -3  1
///5 -1 -gap;
           1 -3  0 -2 -3 -2 -3  8 -2 -4 -4 -2 -2 -3 -1  0 -2 -2 -3 -4 -1
///-2 -1 -gap;
          -2  0  1  0 -4  0  0 -2 13 -3 -2 -1  1 -2 -2 -1 -2 -5  2 -4  0
///0 -1 -gap;
          -1 -3 -2 -4 -4 -3 -4 -4 -3  6  2 -3  1  1 -2 -2 -1 -3  0  4 -3
///-4 -1 -gap;
          -2 -2 -3 -3 -2 -2 -2 -4 -2  2  6 -2  3  2 -4 -3 -1 -1  0  2 -3
///-2 -1 -gap;
          -1  3  0  0 -3  1  1 -2 -1 -3 -2  6 -1 -3 -1  0  0 -2 -1 -2  0
///1 -1 -gap;
          -1 -1 -2 -3 -3 -1 -2 -2  1  1  3 -1  7  0 -2 -2 -1 -2  1  1 -3
///-2  0 -gap;
          -3 -2 -3 -4 -2 -4 -3 -3 -2  1  2 -3  0  9 -4 -2 -1  1  4  0 -3
///-4 -1 -gap;
          -2 -3 -2 -2 -5 -2  0 -1 -2 -2 -4 -1 -2 -4 11 -1  0 -4 -3 -3 -2
///-1 -2 -gap;
           1 -1  1  0 -1  1  0  0 -1 -2 -3  0 -2 -2 -1  5  2 -5 -2 -1  0
///0  0 -gap;
           0 -2  0 -1 -1 -1 -1 -2 -2 -1 -1  0 -1 -1  0  2  6 -4 -1  1  0
///-1  0 -gap;
          -3 -2 -4 -5 -6 -1 -2 -2 -5 -3 -1 -2 -2  1 -4 -5 -4 19  3 -3 -4
///-2 -2 -gap;
          -2 -1 -2 -3 -4 -1 -2 -3  2  0  0 -1  1  4 -3 -2 -1  3  9 -1 -3
///-2 -1 -gap;
           0 -2 -3 -3 -2 -3 -3 -4 -4  4  2 -2  1  0 -3 -1  1 -3 -1  5 -3
///-3 -1 -gap;
          -1 -1  4  6 -2  0  1 -1  0 -3 -3  0 -3 -3 -2  0  0 -4 -3 -3  5
```

```
///2 -1 -gap;
          -1  0  0  1 -3  4  5 -2  0 -4 -2  1 -2 -4 -1  0 -1 -2 -2 -3  2
///5 -1 -gap;
           0 -1 -1 -1 -2 -1 -1 -1 -1 -1 -1 -1  0 -1 -2  0  0 -2 -1 -1 -1
///-1 -1 -gap;
           -gap -gap -gap -gap -gap -gap -gap -gap -gap -gap -gap -gap
///-gap -gap -gap -gap -gap -gap -gap -gap -gap -gap -gap -gapgap];

       A=1;
       R=2;
       N=3;
       D=4;
       C=5;
       Q=6;
       E=7;
       G=8;
       H=9;
       I=10;
       L=11;
       K=12;
       M=13;
       F=14;
       P=15;
       S=16;
       T=17;
       W=18;
       Y=19;
       V=20;
       B=21;
       Z=22;
       X=23;
       g=24;


       % This is how to make a dialogue box
       % prompt = {'Enter matrix size:','Enter colormap name:'};
       % dlg_title = 'Input for peaks function';
       % num_lines = 1;
       % def = {'20','hsv'};
       % answer = inputdlg(prompt,dlg_title,num_lines,def);
       str1len=length(String1); %gets the lenght of the first sequence
       str2len=length(String2); %gets the lenght of the second sequence
       valmatrix=zeros(str2len+1,str1len+1); %|
       for x=1:str1len                       %|
       valmatrix(1,x+1)=-gap*x;              %|
       end                                   %|creates the matrix and
///populates the first row and column with the gap penatily
       for x=1:str2len                       %|
       valmatrix(x+1,1)=-gap*x;              %|
       end                                   %|
       for y=2:str1len+1
```

```
       for x=2:str2len+1
            if String1(y-1)=='-'
                String1(y-1)='g';
            end
            if String2(x-1)=='-'
                String2(x-1)='g';
            end

           valmatrix(x,y)=max([valmatrix(x-1,y-1)+BSTABLE(eval(String1(y
///-1)),eval(String2(x-1))) valmatrix(x-1,y)-gap valmatrix(x,y-1)-gap]);
           %populates with the max function


       end
       end
       %disp('valmatrix made')
       sequence=zeros(1,max([2*(str1len+1) 2*(str2len+1)])); %creates
///the matrix
       that holds the sequence raw data
       sequence(1)=str2len+1;
       sequence(2)=str1len+1;
       j=str1len+1;
       i=str2len+1;
       n=3;
       d=1;
       prevd=1;
       ns=0;
       prevseq=1;
       finished=1;
       info=[];
       first=1;
       total=0;
       numseq=1;
       while finished == 1
           check=0;  %check allows to check for the several different
///situations
%matrix(i,j) is row i column j
           if i~=1&&j~=1
           if valmatrix(i,j)== valmatrix(i-1,j-1)+BSTABLE(eval(String2
///(i-1)),eval(String1(j-1)))
               check=check+1;
           end
           if valmatrix(i,j)==valmatrix(i-1,j)-gap
               check=check+3;
           end
           if valmatrix(i,j)==valmatrix(i,j-1)-gap
               check=check+5;
           end
           switch check
               case 1 % move diagonal
                   sequence(d,n)=i-1;
```

```
        sequence(d,n+1)=j-1;
        i=i-1;
        j=j-1;
        n=n+2;
    case 3 % move up
        sequence(d,n)=i-1;
        sequence(d,n+1)=j;
        i=i-1;
        n=n+2;
    case 5 % move left
        sequence(d,n)=i;
        sequence(d,n+1)=j-1;
        j=j-1;
        n=n+2;
    case 4 %move up and diagonal
        numseq=numseq+1;
        sequence=[sequence;zeros(1,length(sequence(1,:)))];
        info=[info,i,j,n];
        ns=ns+1;
        prevseq(ns)=d;
        d=numseq;
        for istart=1:n-1
            sequence(d,istart)=sequence(prevseq(ns),istart);
        end
        sequence(d,n)=i-1;
        sequence(d,n+1)=j-1;
        i=i-1;
        j=j-1;
        n=n+2;
        type=1;

        %disp('branch 1')
    case 6 % move left and diagonal
        numseq=numseq+1;
        sequence=[sequence;zeros(1,length(sequence(1,:)))];
        info=[info,i,j,n];
        ns=ns+1;
        prevseq(ns)=d;
        d=numseq;
        for istart=1:n-1
            sequence(d,istart)=sequence(prevseq(ns),istart);
        end
        sequence(d,n)=i-1;
        sequence(d,n+1)=j-1;
        i=i-1;
        j=j-1;
        n=n+2;
        type=2;

        %disp('branch 2')
    case 8 % move up and left
```

```
                    numseq=numseq+1;
                    sequence=[sequence;zeros(1,length(sequence(1,:)))];
                    info=[info,i,j,n];
                    ns=ns+1;
                    prevseq(ns)=d;
                    d=numseq;
                    for istart=1:n-1
                        sequence(d,istart)=sequence(prevseq(ns),istart);
                    end
                    sequence(d,n)=i-1;
                    sequence(d,n+1)=j;
                    i=i-1;
                    n=n+2;
                    type=3;

                    %disp('branch 3')
                case 9 %move all three
                    numseq=numseq+2;
                    sequence=[sequence;zeros(1,length(sequence(1,:)))];
                    info=[info,i,j,n];
                    info=[info,i,j,n];
                    ns=ns+1;
                    prevseq(ns)=d;
                    ns=ns+1;
                    prevseq(ns)=d+1;
                    d=numseq;
                    for istart=1:n-1
                        sequence(d-1,istart)=sequence(prevseq(ns-1),istar
///t);
                    end
                    for istart=1:n-1
                        sequence(d,istart)=sequence(prevseq(ns),istart);
                    end
                    sequence(d,n)=i-1;
                    sequence(d,n+1)=j-1;
                    i=i-1;
                    j=j-1;
                    n=n+2;
                    type=4;

                    disp('branch 4')
            end
            end
            %Fills in the rest of the sequences when the other runs out
            if i==1
                for q=1:j-1
                    sequence(d,n)=i;
                    sequence(d,n+1)=j-1;%d is 4 not 5 it needs to be 5
                    j=j-1;
                    n=n+2;
                end
```

```
            end
            if j==1
                for q=1:i-1
                    sequence(d,n)=i-1;
                    sequence(d,n+1)=j;
                    i=i-1;
                    n=n+2;
                end
            end
            %calculates approx number of sequences left
            if i==1&&j==1&&d~=1
                if first==1
                    total=2^(d-1);
                    %disp(['approximate number of sequences: ' num2str(tot
///al)])
                end
                first=first+1;
                if mod(first,floor(total/100))==0
                    disp(round(first/total*100))
                end
                %deals with branch points
                switch type
                    case 1
                    i=info(3*(d-1)-2);
                    j=info(3*(d-1)-1);
                    n=info(3*(d-1));

                    sequence(prevseq(ns),n)=i-1;
                    sequence(prevseq(ns),n+1)=j;
                    i=i-1;
                    n=n+2;

                    case 2
                    i=info(3*(d-1)-2);
                    j=info(3*(d-1)-1);
                    n=info(3*(d-1));

                    sequence(prevseq(ns),n)=i;
                    sequence(prevseq(ns),n+1)=j-1;
                    j=j-1;
                    n=n+2;

                    case 3
                    i=info(3*(d-1)-2);
                    j=info(3*(d-1)-1);
                    n=info(3*(d-1));

                    sequence(prevseq(ns),n)=i;
                    sequence(prevseq(ns),n+1)=j-1;
                    j=j-1;
                    n=n+2;
```

```
                    case 4
                    i=info(3*(d-1)-2);
                    j=info(3*(d-1)-1);
                    n=info(3*(d-1));

                    sequence(prevseq(ns),n)=i-1;
                    sequence(prevseq(ns),n+1)=j;
                    type = 5;
                    i=i-1;
                    n=n+2;

                    case 5
                    i=info(3*(d-1)-2);
                    j=info(3*(d-1)-1);
                    n=info(3*(d-1));

                    sequence(prevseq(ns),n)=i;
                    sequence(prevseq(ns),n+1)=j-1;
                    j=j-1;
                    n=n+2;

                end
                prevd=d-prevd;
                d=prevseq(ns);
                ns=ns-1;
                %disp('one more sequence finished')
                if d<=0
                    d=1;
                end
            end
            if i==1&&j==1&&d==1
                %disp('all sequences finished')
                finished=0;
            end

        end

        if isempty(info)
            numseq = 0;
        else
            numseq=length(info)/3;
        end
        seq1='';
        seq2='';
        final='';
        seqval=zeros(1,numseq+1);
%       for x=1:numseq+1
%           z=x;
%           len=length(sequence(x,:));
%           seq1='';
```

```
%           seq2='';
%           for y=1:(len-2)/2
%               if z>=2
%                   if y==(len-info(3*(z-1))+3)/2
%                   z=z-1;
%                   end
%               end
% %               if String1(sequence(z,len+2-2*y))=='8'
% %                   String1(sequence(z,len+2-2*y))='-';
% %               end
% %               if String2(sequence(z,len+2-2*y))=='8'
% %                   String2(sequence(z,len+2-2*y))='-';
% %               end
%               if sequence(z,len+2-2*y)==sequence(z,len-2+2-2*y)
%                   seq1=[seq1,'g'];
%                   seq1ref='g';
%               else
%                   seq1=[seq1,String1(sequence(z,len+2-2*y))];
%                   seq1ref=String1(sequence(z,len+2-2*y));
%               end
%               if sequence(z,len+1-2*y)==sequence(z,len-2+1-2*y)
%                   seq2=[seq2,'g'];
%                   seq2ref='g';
%               else
%                   seq2=[seq2,String2(sequence(z,len+1-2*y))];
%                   seq2ref=String2(sequence(z,len+1-2*y));
%               end
%               a=sequence(z,len-1+2-2*y);
%               b=sequence(z,len+2-2*y);
%               seqval(x)=seqval(x)+BSTABLE(eval(seq1ref),eval(seq2ref));
%
%               if seq2(y)=='g'
%                   seq2(y)='-';
%               end
%               if seq1(y)=='g'
%                   seq1(y)='-';
%               end
%           end
%
%           %seqval(x)=seqval(x)+valmatrix(sequence(z,len-1+2-2*(y+1)),se
\\\quence(z,len+2-2*(y+1)));
%           final=char(final,seq1,seq2);
%           disp(['sequence ' num2str(x) ' of ' num2str(numseq+1) ' finis
\\\hed'])
%       end

       for x=1:length(sequence(:,x))
%           z=x;
          len=length(sequence(x,:));
          seq1='  ';
          seq2='  ';
```

```
            for y=1:(len-2)/2
%               if z>=2
%                   if y==(len-info(3*(z-1))+3)/2
%                   z=z-1;
%                   end
%               end
%               if String1(sequence(z,len+2-2*y))=='8'
%                   String1(sequence(z,len+2-2*y))='-';
%               end
%               if String2(sequence(z,len+2-2*y))=='8'
%                   String2(sequence(z,len+2-2*y))='-';
%               end
               if sequence(x,2*y+2)~=0&&sequence(x,2*(y-1)+3)~=0
               if sequence(x,2*y)==sequence(x,2*y+2)
                   seq1=['g',seq1];
                   seq1ref='g';
               else
                   seq1=[String1(sequence(x,2*y+2)),seq1];
                   seq1ref=String1(sequence(x,2*y+2));
               end
               if sequence(x,2*(y-1)+1)==sequence(x,2*(y-1)+3)
                   seq2=['g',seq2];
                   seq2ref='g';
               else
                   seq2=[String2(sequence(x,2*(y-1)+3)),seq2];
                   seq2ref=String2(sequence(x,2*(y-1)+3));
               end
               %a=sequence(x,len-1+2-2*y);
               %b=sequence(x,len+2-2*y);
               if (seq2(2)=='-'&&seq2(1)=='g')||(seq1(2)=='-'&&seq1(1)=='g
\\\')
                   seqval(x)=seqval(x)-multigap;
               else
               seqval(x)=seqval(x)+BSTABLE(eval(seq1ref),eval(seq2ref));
               end
               if seq2(1)=='g'
                   seq2(1)='-';
               end
               if seq1(1)=='g'
                   seq1(1)='-';
               end
               end
           end

           %seqval(x)=seqval(x)+valmatrix(sequence(z,len-1+2-2*(y+1)),
           sequence(z,len+2-2*(y+1)));
           final=char(final,seq1,seq2);
           %disp(['sequence ' num2str(x) ' of ' num2str(numseq+1) ' finish
///ed'])

       end
```

```
       String1;
       String2;
       best=find(seqval==max(seqval(:)));%what if a gap is added to first
///sequence
       FINAL=char(FINAL,final(2*best(1)+1,:));%use best(end) for interest
///ing results
       %,final(2*best(1),:)
       for other=1:length(best)
           final_first=char(final_first,final(2*best(other),:));
           numbest(w)=length(best);
           FINAL_other=char(FINAL_other,final(2*best(other)+1,:));
           test=length(strtrim(final_first(other+1,:)));
           if ~strcmp(strtrim(final_first(other+1,:)),current)
               unique=0;
               if numoversized==0
                   Oversized=char(final_first(other+1,:));
                   numoversized=numoversized+1;
               else
               for ovs=1:numoversized
                   if ~strcmp(strtrim(Oversized(ovs,:)),strtrim(final_fir
///st(other+1,:)))
                        unique=unique+1;
                   else

                   end
               end
               end
               if unique==numoversized
               Oversized=char(Oversized,final_first(other+1,:));
               numoversized=numoversized+1;
               end
               broken=1;
           end
       end
       if broken==1
           if numoversized==1
           current=strtrim(Oversized(1,:));
           %disp('reset')
           return
           else
           disp(Oversized)
           disp('Warning: the length of the reference string has increase
///d. Choose a new reference sequence')
           new=input('New Sequence: ','s');
           current=strtrim(Oversized(str2num(new),:));
           return
           end
       end
%
%       if strcmp(display,'all')==1
```

```
%           for x = 1:numseq+1
%               str=['Sequence alignment #',num2str(x),' is'];
%               str2=final(2*x,:);
%               str3=final(2*x+1,:);
%               str4=['with a score of ' num2str(seqval(x)) '.'];
%               disp(str)
%               disp(str2)
%               disp(str3)
%               disp(str4)
%               disp('    ')
%
%           end
%       end
%       if strcmp(display,'all')~=1
%           [sortedValues,sortIndex] = sort(seqval(:),'descend');
%           maxIndex = sortIndex(1:eval(display));
%           for x = 1:eval(display)
%               str=['Sequence alignment #',num2str(x),' is'];
%               str2=final(2*maxIndex(x),:);
%               str3=final(2*maxIndex(x)+1,:);
%               str4=['with a score of ' num2str(seqval(maxIndex(x))) '.'
///];
%               disp(str)
%               disp(str2)
%               disp(str3)
%               disp(str4)
%               disp('    ')
%
%           end
%       end
%
```

APPENDIX C: MULTIPLE SEQUENCE ALIGNMENT CODE

```
%Seqlist={'MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKK
///VADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLAS
///VSTVLTSKYR','MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMG
///NPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPV
///QAAYQKVVAGVANALAHKYH','MGLSDGEWQLVLNVWGKVEADIPGHGQEVLIRLFKGHPETLEKFDKF
///KHLKSEDEMKASEDLKKHGATVLTALGGILKKKGHHEAEIKPLAQSHATKHKIPVKYLEFISECIIQVLQ
///SKHPGDFGADAQGAMNKALELFRKDMASNYKELGFQG','MGLSDGEWQLVLNVWGKVEADLAGHGQEVL
///ISHRLFKGHPETLEKFDKFKHLKSEDEMKGSEDLKKHGNTVFTALGGILKKKGQHEAELKPLAQSHATKH
///KIPVKYLEFISEAIIQVLQSKHPGDFGADAQAAMRKALELFRNDIAAKYKELGFQG'};
%Seqlist={'MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKK
///VADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLAS
///VSTVLTSKYR','MGLSDGEWQLVLNVWGKVEADLAGHGQEVLISHRLFKGHPETLEKFDKFKHLKSEDE
///MKGSEDLKKHGNTVFTALGGILKKKGQHEAELKPLAQSHATKHKIPVKYLEFISEAIIQVLQSKHPGDFG
///ADAQAAMRKALELFRNDIAAKYKELGFQG'};
%Seqlist={'M-GLSDGEWQLVLNVWGKVEADLAGHGQEVLISHRLFKGHPETLEKFDKFKHLKSEDEMKGS
///EDLKKHGNTVFTALGGILKKKGQHEAELKPLAQSHATK-HKIPVKYLEFISEAIIQVLQSKHPG-DFGAD
///AQAAMRKALELFRNDIAAKYKELGFQG','MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTT
///KTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTL
///AAHLPAEFTPAVHASLDKFLASVSTVLTSKYR'};
%Seqlist={'MAFSLEEESGRIKDCWDNQEVPALSTCSNANIFRRINAILDDSLDFSKVCTTPINRGIHDQL
///PDFQDSEEAITSRMLFPTSAQESPRGLPDANGLCLGLQSLSLTGWDRPWSTQDSDSSAQSNTQSVLSMLQ
///NPLGNVLGKTPLSFLSLDPLGSDLDKFPAPSVRGSRLDTRPILDSRSSSPSDSDTSGFSSGSDHLSDLIS
///SLRISPPLPFLSMTGNGPRDPLKMGVGSRMDQEQAALAAVAPSPTSAPKRWPGTSVWPSWDLLGAPKDPF
///SIEREARLHRQAAAVNEATCTWSGQLPPRNYKNPIYSCKVFLGGVPWDITEAGLVNTFRVFGSLSVEWPG
///KDGKHPRCPPKGNMPKGYVYLVFELEKSVRALLQACSHDPLSPDGLSEYYFKMSSRRMRCKEVQVIPWVL
///ADSNFVWSPSQRLDPSRTVFVGALHGMLNAEALAAILNDLFGGVVYAGIDTDKHKYPIGSGRVTFNNQRS
///YLKAVTAAFVEIKTTKFTKKVQIDPYLEDSLCLICSSQPGPFFCRDQVCFKYFCRSCWHWRHSMEGLRHH
///SPLMRNQKN','MALSLEEEAGRIKDCWDNQEAPALSTCSNANIFRRINAILDNSLDFSRVCTTPINRGI
///HDHLPDFQDSEETVTSRMLFPTSAQESSRGLPDANDLCLGLQSLSLTGWDRPWSTQDSDSSAQSSTHSVL
///SMLHNPLGNVLGKPPLSFLPLDPLGSDLVDKFPAPSVRGSRLDTRPILDSRSSSPSDSDTSGFSSGSDHL
///SDLISSLRISPPLPFLSLSGGGPRDPLKMGVGSRMDQEQAALAAVTPSPTSASKRWPGASVWPSWDLLEA
///PKDPFSIEREARLHRQAAAVNEATCTWSGQLPPRNYKNPIYSCKVFLGGVPWDITEAGLVNTFRVFGSLS
///VEWPGKDGKHPRCPPKGNMPKGYVYLVFELEKSVRSLLQACSHDPLSPGLSEYYFKMSSRRMRCKEVQVI
///PWVLADSNFVRSPSQRLDPSRTVFVGALHGMLNAEALAAILNDLFGGVVYAGIDTDKHKYPIGSGRVTFN
///NQRSYLKAVSAAFVEIKTTKFTKKVQIDPYLEDSLCHICSSQPGPFFCRDQVCFKYFCRSCWHWRHSMEG
///LRHHSPLMRNQKNRDSS'};
Seqlist={'MAFSLEEESGRIKDCWDNQEVPALSTCSNANIFRRINAILDDSLDFSKVCTTPINRGIHDQLP
///DFQDSEEAITSRMLFPTSAQESPRGLPDANGLCLGLQSLSLTGWDRPWSTQDSDSSAQSNTQSVLSMLQN
///PLGNVLGKTPLSFLSLDPLGSDLDKFPAPSVRGSRLDTRPILDSRSSSPSDSDTSGFSSGSDHLSDLISS
///LRISPPLPFLSMTGNGPRDPLKMGVGSRMDQEQAALAAVAPSPTSAPKRWPGTSVWPSWDLLGAPKDPFS
///IEREARLHRQAAAVNEATCTWSGQLPPRNYKNCKVFLGGVPWDITEAGLVNTFRVFGSLSVEWPGKDGKH
///PRCNMPKGYVYLVFELEKSVRALLQACSHDPLSPDGLSEYYFKMSSRRMRCKEVQVIPWVLADSNFVWSP
///SQRLDPSRTVFVGALHGMLNAEALAAILNDLFGGVVYAGIDTDKHKYPIGSGRVTFNNQRSYLKAVTAAF
///VEIKTTKFTKKVQIDPYLEDSLCLICSSQPGPFFCRDQVCFKYFCRSCWHWRHSMEGLRHHSPLMRNQKN
///','MALSLEEEAGRIKDCWDNQEAPALSTCSNANIFRRINAILDNSLDFSRVCTTPINRGIHDHLPDFQD
///SEETVTSRMLFPTSAQESSRGLPDANDLCLGLQSLSLTGWDRPWSTQDSDSSAQSSTHSVLSMLHNPLGN
///VLGKPPLSFLPLDPLGSDLVDKFPAPSVRGSRLDTRPILDSRSSSPSDSDTSGFSSGSDHLSDLISSLRI
///SPPLPFLSLSGGGPRDPLKMGVGSRMDQEQAALAAVTPSPTSASKRWPGASVWPSWDLLEAPKDPFSIER
```

```
///EARLHRQAAAVNEATCTWSGQLPPRNYKNPIYSCKVFLGGVPWDITEAGLVNTFRVFGSLSVEWPGKDGK
///HPRCPPKGNMPKGYVYLVFELEKSVRSLLQACSHDPLSPGLSEYYFKMSSRRMRCKEVQVIPWVLADSNF
///VRSPSQRLDPSRTVFVGALHGMLNAEALAAILNDLFGGVVYAGIDTDKHKYPIGSGRVTFNNQRSYLKAV
///SAAFVEIKTTKFTKKVQIDPYLEDSLCHICSSQPGPFFCRDQVCFKYFCRSCWHWRHSMEGLRHHSPLMR
///NQKNRDSS'};
%^stay away from gap<2 but gap=40000 is interesting
%Seqlist={'FTFTALILLAVAV','FTALLLAAV','FTFALIAV'}; %affine
%Seqlist={'FTALLLAAV','FTFTALILLAVAV','FTFALIAV','FTALAV'};
%Seqlist={'KTEAEMKASEDLKKHGT','KT'};
%Seqlist={'KTEAEMKASEDLKKHGT','HGSAQVKGHG','AEMKGHG'}; %try with gap=1
%Seqlist={'WWWWWWWC','WWWWWWWW'}; %with gap=8 gap=0 the reason gap doesnt
                                  %have much of an affect is because the
                                  %blosome table values are two low. once
                                  %gap is higher than a certain value it
                                  %become lest costly to swap than gap. 8
                                  %seems to be that number. the max
                                  %penalty in BStable is switch from D to
                                  %T for -6. having gap be 8 means that
                                  %is is better to align D with T than
                                  %place a gap. once it is high enough,
                                  %higher doesnt affect the out come
                                  %other than the final score

           %allign with current or first
           %how to pick which one alignment to go with

gap=10;    %when gap is high enough it will force the alignment to be the
%length of the longest string and wont change any more
multigap=5; %affine

display='all';
broken=1;
%right now they have to be arranged by length

for i=1:length(Seqlist)
    strlengths(i)=length(char(Seqlist(i)));
end
[junk,sorted]=sort(strlengths,'descend');
current=char(Seqlist(sorted==1));
FINAL=char(Seqlist(sorted==1));
next=1;
while broken==1
    Oversized='';
    numoversized=0;
    FINAL_other='';
    final_first='';
    numbest=zeros(1,length(Seqlist)-1);
    strlengths=zeros(1,length(Seqlist));
    broken=0;
for w=1:length(Seqlist)-1
final_first='';
```

```
next=find(sorted==(w+1));
seqalign1   %calls the original program for pairwise alignments
%disp('one set done')
if broken==1
    break
end
%current=char(final(3,:));
end
end
%check length if longer that starting string then restart with this strin
///g
multiscore=zeros(1,prod(numbest));
startnum=zeros(1,(length(Seqlist)-1));
startnum(1)=1;
%currentnum=zeros(1,length(best));
multiscorecount=zeros(prod(numbest),(length(Seqlist)-1));
   for num=2:(length(Seqlist)-1)
       startnum(num)=startnum(num-1)+numbest(num-1);
   end

currentnum=startnum;
for fin=1:prod(numbest)
    inc=(length(Seqlist)-1);
    currentnum;
    done=0;
    final_sev=char(current);
    for creat=1:(length(Seqlist)-1)
        final_sev=char(final_sev,FINAL_other(currentnum(creat)+1,:));
    end


%%%%%%%changing order still results in same max
%     final_sev=char(final_sev,FINAL_other(currentnum(2)+1,:),FINAL_other
///(currentnum(1)+1,:),
FINAL_other(currentnum(3)+1,:));
%     final_sev;
    %%%%%interesting when n is replace with 1
    for n=1:length(Seqlist)-1
        for i=1:length(strtrim(final_sev))
            %if final_sev(n,i)~=' '&&final_sev(n+1,i)~=' '
            if final_sev(n,i)=='-'
                AM1=24;
            else
                AM1=eval(final_sev(n,i));
            end
            if final_sev(n+1,i)=='-'
                AM2=24;
            else
                AM2=eval(final_sev(n+1,i));
            end
            if i>=2&&((final_sev(n,i)=='-'&&final_sev(n,i-1)=='-')||(fina
```

```
///l_sev(n+1,i)==’-’&&final_sev(n+1,i-1)==’-’))&&final_sev(n,i)~=final_se
///v(n+1,i)
                multiscore(fin)=multiscore(fin)-multigap;
            else
                multiscore(fin)=multiscore(fin)+BSTABLE(AM1,AM2);
            end
            %end
        end
    end


    %%%%%for all%%%%%%%
     disp([’Multiple Sequence Alignment #’,num2str(fin),’ is’])
     disp(final_sev)
     disp([’with a score of ’ num2str(multiscore(fin))])
     disp(’   ’)
    multiscorecount(fin,1:(length(Seqlist)-1))=currentnum;
        if currentnum(inc)==sum(numbest(1:inc))
            currentnum(inc)=startnum(inc)-1;
            inc=inc-1;
            while done==0&&inc~=0
                if numbest(inc)~=1
                    currentnum(inc)=currentnum(inc)+1;
                    done=1;
                else
                    inc=inc-1;
                end
            end
        end

    currentnum((length(Seqlist)-1))=currentnum((length(Seqlist)-1))+1;

end

%%%%%%%%%%For Maxs only%%%%%%%
% [maxx,loc]=find(max(multiscore)==multiscore);
% for fin=1:length(loc)
% final_sev=char(Seqlist(sorted==1));
%     for creat=1:length(best)
%         final_sev=char(final_sev,FINAL_other(multiscorecount(loc(fin),c
///reat)+1,:));
%     end
%
%      disp(’Best Multiple Sequence Alignment’)
%      disp(final_sev)
%      disp([’with a score of ’ num2str(multiscore(loc(fin)))])
%
% end

%%%%%%%%%%%%for fisrt only%%%%%%%%%
% for n=1:length(Seqlist)-1
```

```
% for i=1:length(FINAL)-2
%     if FINAL(n,i)=='-'
%     AM1=24;
%     else
%     AM1=eval(FINAL(n,i));
%     end
%     if FINAL(n+1,i)=='-'
%     AM2=24;
%     else
%     AM2=eval(FINAL(n+1,i));
%     end
%      if i>=2&&((FINAL(n,i)=='-'&&FINAL(n,i-1)=='-')||(FINAL(n+1,i)=='-'
///&&FINAL(n+1,i-1)=='-'))&&FINAL(n,i)~=FINAL(n+1,i)
%          multiscore=multiscore-multigap;
%      else
%     multiscore=multiscore+BSTABLE(AM1,AM2);
%     end
% end
% end
% disp('Final Multiple Sequence Alignment is')
% disp(FINAL)
% disp(['with a score of ' num2str(multiscore)])
```