MAVIS: VISUALIZATION TOOL FOR MULTIPLE SEQUENCE ALIGNMENTS

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Lei Zhao

August 2011

MAVIS: VISUALIZATION TOOL FOR MULTIPLE SEQUENCE ALIGNMENTS

Lei Zhao

APPROVED:

Dr. Yuriy Fofanov, Chairman
Department of Computer Science

Dr. Dan Graur
Department of Biology and Biochemistry

Dr. Shishir Shah
Department of Computer Science

Dean, College of Natural Sciences and Mathematics

Acknowledgements

I am very grateful to my thesis supervisors, Dr. Dan Graur and Dr. Reed A. Cartwright, for their guidance, encouragement, patience, and support during the course of this work. I have encountered may difficulties, but your help made this thesis possible.

I would like to thank Dr. Yuriy Fofanov and Dr. Shishir Shah for being my thesis committee members and providing valuable suggestions.

I would like to thank my lab mates, Dr. Giddy Landan, Dr. Kiyoshi Izawa, Dr. Niv Sabath, Nicholas Price, Longyi Qi, and Yichen Zheng. I had a great time in this lab for two years and learned a lot.

I would like to thank my other friends in the University of Houston for all their support, entertainment, and caring.

My special thank goes to my wife Bingjun Zhang, my parents Dr. Wenyuan Zhao and Yijun Wang, and my previous supervisor when I was in China, Dr. Kui Lin. They are the most important persons in my life. To them I dedicate this thesis.

MAVIS: VISUALIZATION TOOL FOR MULTIPLE SEQUENCE ALIGNMENTS

An Abstract of a Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Lei Zhao

August 2011

Abstract

Biological sequences like DNA, RNA, and protein are frequently analyzed in the field of molecular biology and bioinformatics. Multiple sequence alignments (MSAs) help identify similarity between the sequences and reveal their functional, structural, or evolutionary relationships. MSA analysis often requires various visualization techniques, which assist researchers to better understand, evaluate, and learn from the MSA results. One of the common techniques is the coloring of the MSA. There are several coloring tools available, but their color schemes focus more on chemical properties of the sequence residues, rather than the actual structural or evolutionary information of the entire alignment.

In this thesis, we introduce Mavis, a new approach to coloring MSAs and highlighting its quality and structure. Instead of using a pre-defined color scheme based on residue types, we designed a new algorithm to dynamically generate a color for each residue based on a user-determined similarity score between this residue and others. This new tool colors an MSA in such a way that a well aligned region is represented by a solid-color block, while a poorly aligned one by a mosaic with various colors. Thus, the alignment quality and internal structure are clearly displayed.

We implemented the core Mavis algorithm in R, the application programming interface (API) in Perl CGI, and the graphical user interface (GUI) in HTML and

JavaScript that can be viewed over web browsers. Mavis is ready-to-use for biologists without requiring advanced computer science skills and will deliver the visualization result of a typical MSA in minutes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Biological sequences, including nucleotide sequences (DNA and RNA) and peptide sequences (proteins), play an fundamental role in life and reproduction. A nucleotide sequence consists of a chain of linked nucleotide bases. There are four possible types of bases in a DNA sequence, represented by four letters A, T, G, and C. A peptide sequence consists of linked amino acids, which have 20 different common types. Figure 1.1 and 1.2 shows their notation rules recommended by the International Union of Pure and Applied Chemistry (IUPAC) [1].

Since the 1970s, sequences of more than 160,000 organisms have been decoded and analyzed by researchers [2]. With the explosively growing amount of data, it is impractical to study sequences manually. Computer-aided techniques are needed to

Table 1.1: IUPAC Nucleotide Notation Rules

IUPAC Code	Base
A	Adenine
C	Cytosine
G	Guanine
T (or U)	Thymine (or Uracil)
N	any base
. or -	gap

help analyze their structures, functions and evolution.

Multiple sequence alignment (MSA) is one of the computational ways to identify regions of similarity between sequences and is required by almost all comparative sequence studies. An MSA is typically organized as a matrix, in which each row represents a sequence, and each column corresponds to an hypothetically equivalent position across all the sequences aligned. Each element of the matrix contains a symbol, which can be either '-' (a gap) or a sequence symbol (a nucleotide base for DNA/RNA sequences, or an amino acid for protein sequences) [3].

```
AY169803.O  MHYRDLSTLIIVSALLLINVXLWMFILRXYLEHKRQERREREILERLRRIREIKDDSDYESNGEEEQEVMD-LVHSHGFDNPMFEL
AY169809.O  MQYKGL--LLIIIALLLINVXVWMFNLRKYLEQKKQERREREVINRLRRIREVKDDSDYESNGEEEQEVME-LVHSHGFDNPMFEL
AY169807.O  MLHRDLLLLIIISALLLTNIILWMFVLRKYLEIKKQERREREILERLRXIREIRDDSDYESNEEEEQEVRDHLVHTFGFANPMFEI
AY169811.O  MHHRDLLTLIAVSALLFINIILWIYVLRKYLEQRKQDRREREILERLRRIXEIGDDSDYESNEEEEQEVMD-LVHNHGFDNPMFEP
AJ302646.O  MHHRDLLALITTSALLLTNVVLWTFILRQYLKQKKQDKREREILERLRRIRQIEDDSDYESDGTEEQEVRD-LVHSYGFDNPMFEL
AY169815.O  MQHKDLLILIITSALLLINVILWLFVLKQCLEQKKQTKREREIIRRLRRIREIEDDSDYESNGEEEQTVRD-LIHSHGFDNPMFEL
AY169804.O  MHQRDLLILIAVSILCLICILVWTFNLRKYLEHRKQDKREREILERLRRVREIRDDSDYESBGEEEQEVMD-LIHSHGFANPLFEL
AY169810.O  MNYKELLSLIVVSVLLLAAIVIWMFILKKYLEQKEQDRRERELLKRIERLXEXRDDSDYESNGDEEQEVMH-LVHTHGFANPMFEL
AY169806.O  MYKDQIILIIIFCVVFLIAACIWLFILKTYLEQKKQDRREKELLRRLQRIIEIRDDSDYESNGEEEQEVMD-LVHEHGFVNPMFEL
```

Figure 1.1: An example of multiple sequence alignment, generated by *ClustalW* [4]. Each row in the MSA represents one protein sequence. The leftmost column shows sequence names. The right part is the alignment matrix.

An MSA result can be a large and complex matrix. To highlight similarity or other properties, certain visualization methods are applied. Some alignment programs, like

Table 1.2: IUPAC Amino Acid Notation Rules

IUPAC Code	Three Letter Code	Amino acid
A	Ala	Alanine
C	Cys	Cysteine
D	Asp	Aspartic Acid
E	Glu	Glutamic Acid
F	Phe	Phenylalanine
G	Gly	Glycine
H	His	Histidine
I	Ile	Isoleucine
K	Lys	Lysine
L	Leu	Leucine
M	Met	Methionine
N	Asn	Asparagine
P	Pro	Proline
Q	Gln	Glutamine
R	Arg	Arginine
S	Ser	Serine
T	Thr	Threonine
V	Val	Valine
W	Trp	Tryptophan
Y	Tyr	Tyrosine

ClustalW, add an asterisk, colon, semicolon, or other symbols, to show conservative columns. Some others use color to display information: assign each nucleotide or amino acid its own color, and indicate amino acid properties with an empirical color assignment. Figure 1.2 shows a protein MSA colored using Jalview [5].

1.2 Related Work

Numerous MSA visualization applications, either stand-alone or web-based, have been developed in recent decades, ranging from simple MSA viewers, to complex editing and analysis platforms. Many of them have graphical user interfaces showing

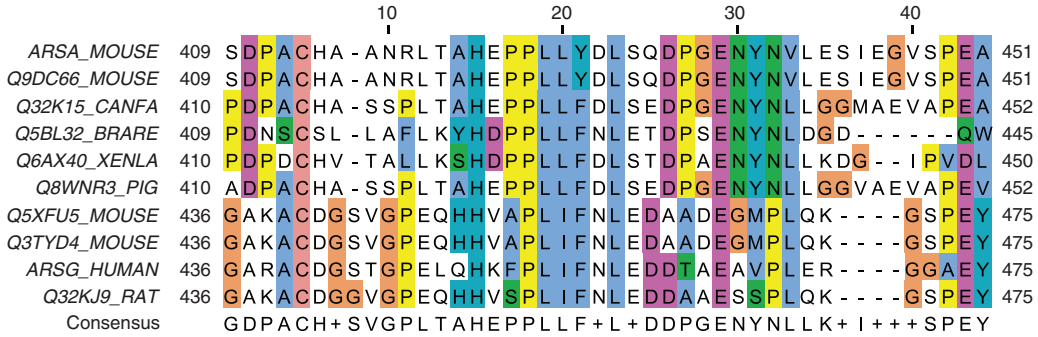


Figure 1.2: A protein MSA colored with Jalview [5, 6].

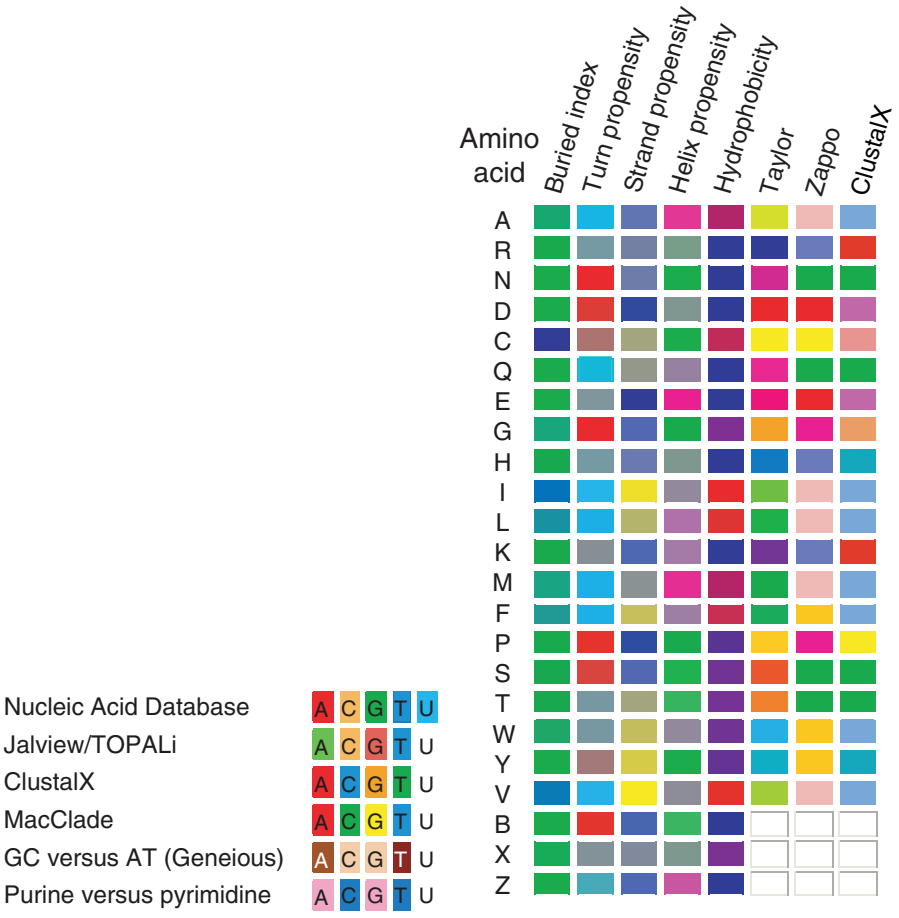
a grid of symbols, usually along with various colorings, sizings, and annotations [6].

Coloring an alignment has been widely implemented to highlight specific regions, properties, or patterns. For example, sequences can be colored according to the similarity to consensus sequence, physicochemical properties of amino acids, or secondary structure of the proteins.

The simplest way of coloring is to use a fixed color scheme, in which each sequence symbol has its own color, and will not be changed across the alignment. The assignment of color schemes are usually based on some empirical properties or chemical classifications. Figure 1.3 shows some nucleotide and amino acid color schemes used by major visualization tools.

Numerous color schemes are available for different purposes. Taylor [7] and Clustal [4] are two of the most frequently used and often considered *de facto* standards. Figure 1.4 shows Taylor's Venn diagram of color scheme.

Using these pre-determined color schemes to color sequence symbols, however, is



(a) Some nucleotide color schemes. **(b)** Some amino acid color schemes.

Figure 1.3: Examples of color schemes used by some visualization tools [6].

not always helpful for phylogenetic analysis. Those fixed color schemes reflect chemical differences between specific types of symbols, rather than between sequences, regions, blocks, or structures of the alignment.

When symbols are examined within one column, the colors successfully show the similarities. But in the horizontal aspect, comparative information within a row or a rectangular region are unable to be displayed. For example, how is one sequence

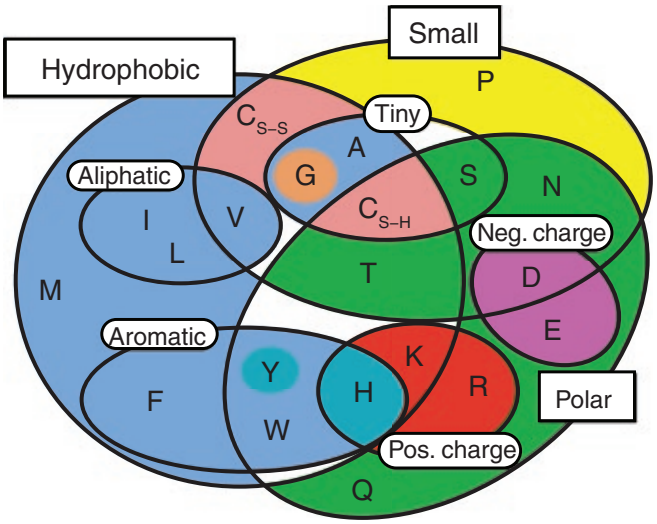
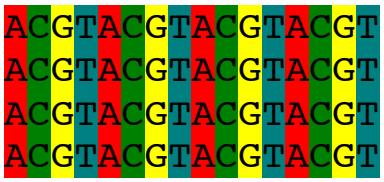


Figure 1.4: Taylor's [7] Venn diagram showing the color scheme based on the physicochemical properties of the amino acid groups [6].

similar to others? Which sections align better than others? Which areas of the alignment are more conserved? How robust is the alignment accuracy?

To make things worse, fixed color scheme can introduce confusions. Color difference between different columns gives an impression of dissimilarity, while it actually represents nothing. Figure 1.5 shows an example of this confusion.



(a) A perfectly aligned MSA. However, the color graph is distractive and fails to convey a 'perfect' feeling.

(b) This is an extremely poorly aligned one, but the color graph looks even better than (a).

Figure 1.5: Fixed color scheme introduces confusions.

1.3 Thesis Outline

In this report, we introduce a new MSA visualization tool called *Mavis* (Multiple Alignment VISualization). Mavis focuses on highlighting evolutionary relationships and internal structures of an alignment. Instead of relying on chemical properties or any other information from specific symbol types, our approach is based on only one metric, which is the symbol-symbol similarity score. The color of a sequence residue will not depend on its symbol type, but on its similarity scores with other residues. By applying this new approach, we are able to provide a big picture of an MSA, in which the structure and quality are clearly displayed.

The rest part of this thesis is organized into five chapters. Chapter 2 explains the general idea of how Mavis is designed and the steps it takes to color an MSA on a high level. Chapter 3 expounds the core algorithm step by step, from multidimensional scaling, to color mapping, and to optimization. Chapter 4 describes the implementation of the algorithm, including the back-end, API, and different user interfaces. Chapter 5 presents some test results to show the validity and performance of our implementation. The entire study is discussed in Chapter 6 and concluded in Chapter 7.

The source code of Mavis back-end, API, and web interface is available in the Appendix at the end of this report.

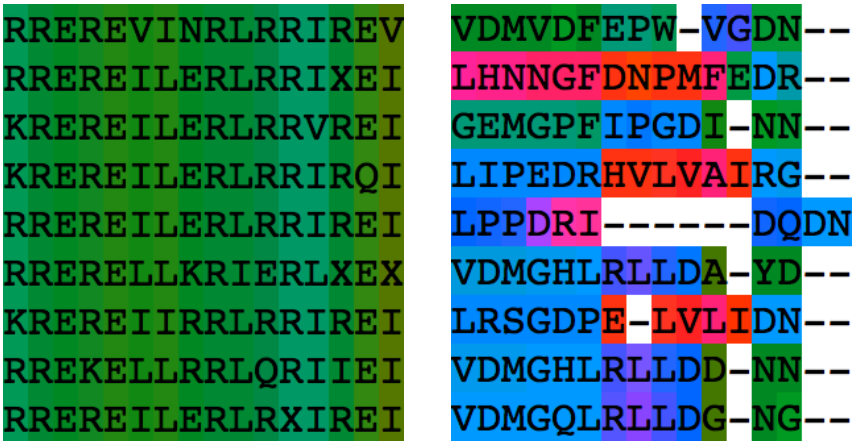
Chapter 2

Approach

2.1 Main Objective

In this thesis, we address a novel method to visualize multiple sequence alignments using colors. The main objective is to highlight sequence similarity (or alignment confidence) with color similarity.

In the alignment matrix, a well-aligned area should be represented by a solid-color shape, while a poorly-aligned one should become a mosaic of different colors. All the alignment-level information, such as evolutionary relationships, phylogenetic characteristics and alignment accuracy, should be revealed by the color pattern of the graph. Figure 2.1 compares these two types of MSA regions and their coloring results.



(a) A well-aligned MSA region. **(b)** A poorly-aligned MSA region.

Figure 2.1: Highlighting sequence similarity with color similarity.

2.2 Data Input

Figure 2.1(a) shows a well-aligned MSA region, which contains many different types of amino acids. In order to demonstrate the good alignment quality, however, these amino acids are supposed to be painted with the same or similar colors. Obviously, the color assignment could not depend only on their residue types.

Therefore, unlike most traditional coloring approaches, we use a set of residue-residue similarity scores instead. Within each column of an MSA, all sequence residues are considered to be aligned together at the same equivalent position. We employ a third-party tool to examine each pair of these residues, and estimate the confidence that they should be aligned together. This confidence estimation can also be interpreted as a similarity score between the two residues.

Every pair of residues within a same column has a similarity score. These scores are used to determine how similar the residue colors should appear to each other. In

a good-quality MSA region, all residues are confidently aligned together, therefore the colors would be same or similar. While in a bad-quality region, many alignments are doubtable, resulting in many different colors.

One widely used tool to generate this similarity score data set is GUIDANCE [8, 9], which is designed for quantifying alignment uncertainty. GUIDANCE produces a confidence score in range of $[0, 1]$, named GUIDANCE score, for each symbol, symbol pair, column and sequence in the alignment. We take the symbol-pair GUIDANCE confidence scores as the input to Mavis.

2.3 Color Generating

The colors of the symbols are generated in such way that the color similarity represents the residue's alignment confidence. We have three steps to accomplish this goal.

First, the confidence or similarity score is converted into a distance or dissimilarity score. From all these distances, an $N \times N$ distance matrix is constructed for each column, where N is the number of residues in the same column. Second, from the distance matrix, the N residues are mapped as N points in a space of $N - 1$ dimensions, such that for each pair of points, their Euclidean distance is equal to the value in the distance matrix. This mapping process is done by using a statistical technique called *classical multidimensional scaling* [10]. Third, we map this space onto a color space, to find a color for each point. In an appropriate color space, the Euclidean distance approximately represents the color difference perceived by

$$\begin{bmatrix} & A & B & C & D \\ A & 0.00 & 0.42 & 0.64 & 0.70 \\ B & 0.42 & 0.00 & 0.81 & 0.57 \\ C & 0.64 & 0.81 & 0.00 & 0.50 \\ D & 0.70 & 0.57 & 0.50 & 0.00 \end{bmatrix}$$

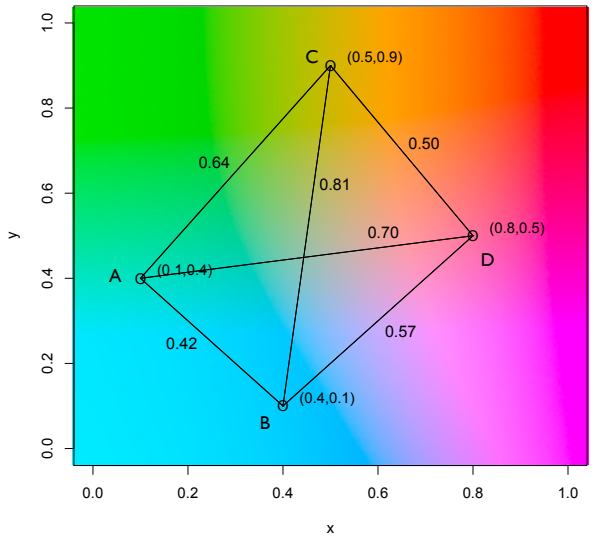


Figure 2.2: A distance matrix is converted into points on a two-dimensional color space. The distances between points are approximately preserved. The color where a point is located is assigned to the corresponding alignment residue.

humans, so that our alignment confidence is linked with color similarity.

One problem appeared in the last step is that most color spaces are three-dimensional. In order to map the original space to a color space, it is necessary to reduce the dimension from $N - 1$ to no more than 3, while the distances are still preserved as much as possible. Fortunately, classical multidimensional scaling is designed in such a way that the result has the largest possible variance on the first axis, the second largest on the second axis, and so on. Thus, we can reduce the space dimension by picking the first a few axes, and mostly preserve the distances.

2.4 Color Optimization

Up to now, our approach only operates on the column level and does not consider the relationship between columns. The only way to create a solid-color well-aligned region is to take cross-column information into account. Since there is no confidence score for two residues from different columns, the color difference between them does not mean anything, and should be eliminated as much as we can. In other words, the color graph needs to be as smooth as possible in the horizontal direction.

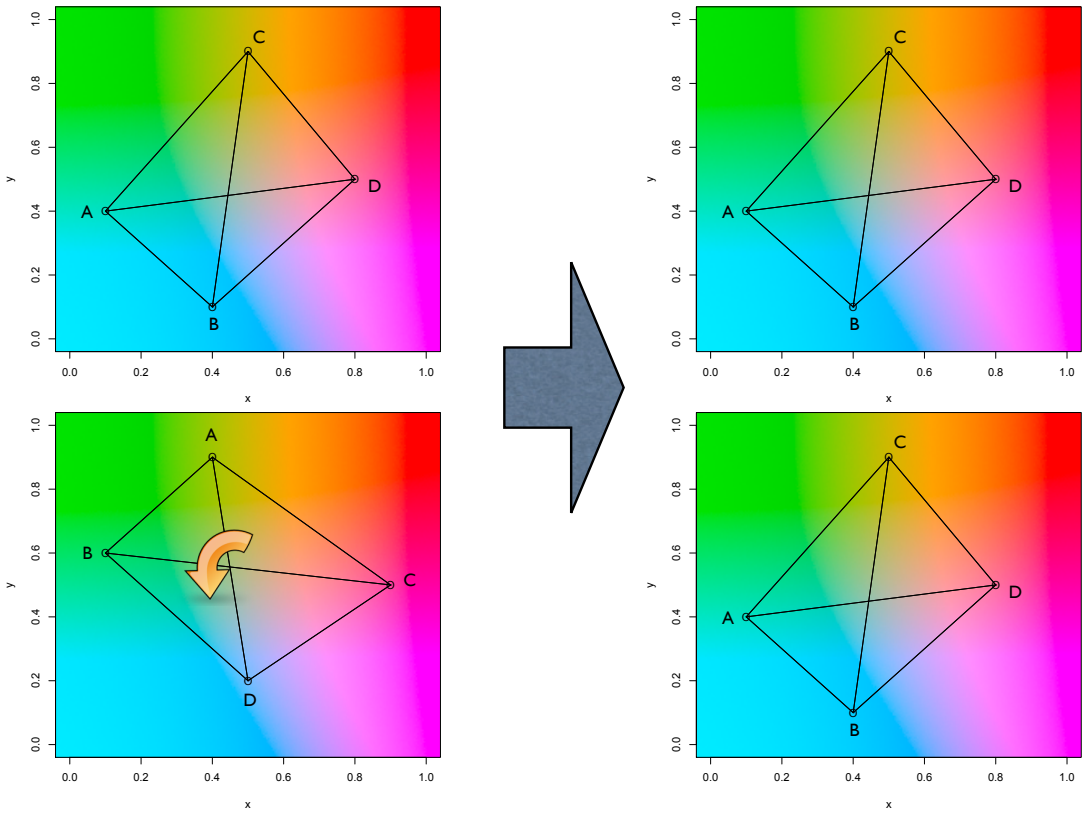


Figure 2.3: An example of color optimization by rotating.

A favorable feature of classical multidimensional scaling is the invariance under flips and rotations. All the points in the color space can be arbitrarily rotated and/or flipped, without changing the distance between any pair of symbols within a column. For instance, all reds and all greens are virtually identical in the sense of distances and quality of alignment. Converting them into the same color pattern will better support this idea and remove unnecessary color noises.

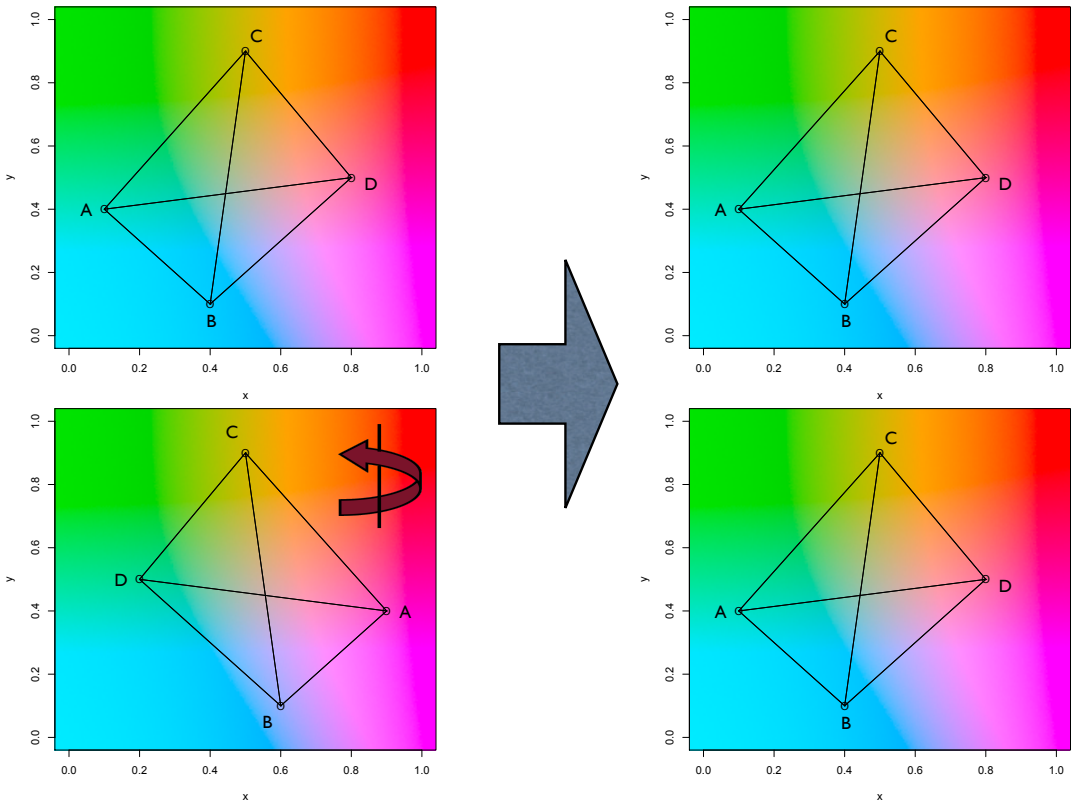


Figure 2.4: An example of color optimization by flipping.

To optimize our color assignment, we define a penalty function to quantify the

overall color noise level of an alignment. This function calculates the color variance within each row and sum them up. By rotating and flipping, the ideal color assignment should result in the lowest penalty value.

For larger MSAs, finding the exact optimum may become time-consuming and sometimes impractical. Sub-optimal results are also acceptable for visualization purpose. In Mavis, we use an approximate solution to speed up the optimization procedure.

2.5 MSA Visualization

Mavis creates a colored matrix for an MSA. Users may choose to display residue symbols or not. The user interface is available in a web browser, in order to achieve better flexibility and portability. An API between the front-end and the back-end is designed to make the system extensible.

Figure 2.5 shows an example of MSA matrix colored by Mavis.

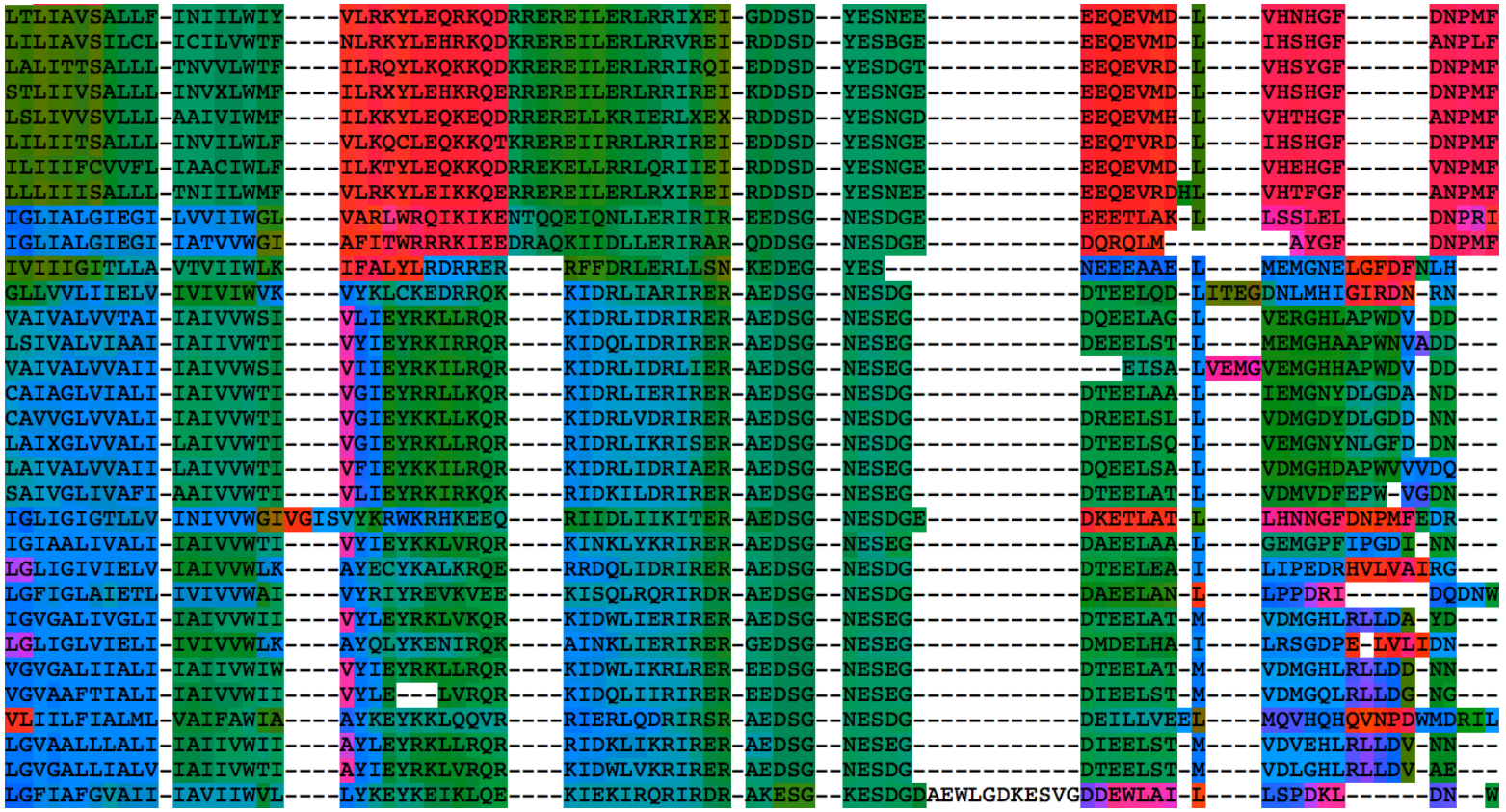


Figure 2.5: An example of MSA matrix colored by Mavis.

Chapter 3

Algorithms

Mavis' algorithm of color computation consists of four stages. First, multidimensional scaling converts the distance matrix into a set of coordinates in a high dimensional space and reduces the number of dimensions to two by selecting the first two axes. Second, coordinates are mapped onto part of the *CIE Lab color space* [11], and colors are created. Third, color spaces for each column is flipped and rotated to smooth the graph and remove color noises. Finally, the average color and average hue of each sequence are calculated, sorted, and output for further display.

We will discuss all these stages step by step in this chapter.

3.1 Multidimensional Scaling

This stage begins with the similarity scores described above. Each column in the MSA has a complete set S of pairwise scores. Each pair of symbols i, j in this column has a score $s_{i,j} = s_{j,i} \in [0, 1]$. Here 1 stands for absolute confidence or similarity, while 0 for the opposite.

This dataset can be easily converted to a set D of dissimilarities or distances between two symbols, by

$$d_{i,j} = d_{j,i} = 1 - s_{i,j}$$

These distance values are re-organized as an $n \times n$ symmetric distance matrix M

$$\begin{array}{cccccc} & 1 & 2 & 3 & ... & n \\ 1 & 0 & d_{1,2} & d_{1,3} & \cdots & d_{1,n} \\ 2 & d_{2,1} & 0 & d_{2,3} & \cdots & d_{2,n} \\ 3 & d_{3,1} & d_{3,2} & 0 & \cdots & d_{3,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ n & d_{n,1} & d_{n,2} & d_{n,3} & \cdots & 0 \end{array}$$

where n is the number of symbols in the current column.

In most cases, gap symbols are removed because they are not compared and present in the similarity data. Therefore n is actually the number of non-gap symbols in the current column. In certain circumstance, however, gaps can also be treated as normal symbols, have similarity scores, and show their colors.

The distance matrix M is then converted into n coordinates

$$C_1, C_2, ..., C_{n-1}$$

$$\text{where } C_i = (c_{i,1}, c_{i,2}, ..., c_{i,n-1})$$

in an $(n-1)$-dimensional space. This procedure involves a statistical technique called *classical multidimensional scaling (classical MDS)*, also known as *Torgerson-Gower scaling* or *principle coordinates analysis (PCoA)* [12].

Classical multidimensional scaling is designed in such a way that the coordinates have the largest variance on the first axis. In other words,

$$Var(c_{1,1}, c_{2,1}, ..., c_{n,1})$$

has the largest possible value in the scaling result. The second axis has the second largest variance, and so on. That means if we pick only first m axes and form a new m-dimensional space, the distances between coordinates are still preserved to a large extent.

These n symbols are placed into a two-dimensional space, in which the distances between each other are still preserved as much as possible. Multidimensional scaling takes a distance matrix and assigns a coordinate to each item in a $(n-1)$-dimensional space, such that the Euclidean distance between any two coordinates is approximately equal to the original distance value between the corresponding items.

An statistical function called *cmdscale()* [13] is utilized in this multidimensional scaling procedure. This function is written in R, which is a programming language for statistical computing [14], and is provided in Rs stat package. It reads a distance

matrix and returns a set of points in a k-dimensional space, where k is a user-defined parameter to the function and must be less than the number of points n [15, 16].

In our approach, k is set to 2, meaning the distance matrix is scaled down to a two-dimensional space. However, a column of an MSA may have as few as one or two non-gap symbols, so that the distance matrix of this column is only 1×1 or 2×2. In such cases, $k = 2$ will not be a valid parameter to *cmdscale()* because of the $k < n$ requirement. So we let

$$k = \begin{cases} 0 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ 2 & \text{otherwise} \end{cases}$$

Then we perform the multidimensional scaling on the distance matrix M. When $n < 2$, the generated coordinates are lower than two-dimensional, and the missing coordinate values are filled by zeros.

3.2 Color Space Mapping

The above procedure returns a set P of n coordinates (x, y) in a two-dimensional space, representing the n symbols in a column of a multiple sequence alignment. This two-dimensional space is further mapped to a three-dimensional color space.

The CIE Lab color space [11] is a color-opponent space. Compared to the *RGB* and *CMYK* color models, the design of Lab color emphasizes more on approximating human vision rather than physical devices. This feature makes it suitable for our

visualization purpose [17].

The name *Lab* stands for the three dimensions: L for lightness, and a and b for the color opponents based on nonlinearly compressed *CIE XYZ color space* coordinates [18, 19]. However, coordinates in P have only two dimensions x and y. To map P to the three-dimensional CIE Lab space, L for lightness is set to a fixed value 75, and two other dimensions a and b are assigned as x and y scaled from $[0, 1]$ to $[0, 100]$.

$$a = x * 100$$

$$b = y * 100$$

At the end of this step, n CIE Lab colors (L, a, b) are created.

3.3 Hue Rotation and Optimization

One of the limitations in many previous MSA visualization techniques is that they introduced unnecessary confusion by using a fixed color scheme. Even for exactly same aligned columns, colors may be totally different, which will not represent any biological dissimilarities. To minimize this effect, we convert the colors from CIE Lab space to *CIE LCH space* and perform rotations to eliminate noisy color patterns to the greatest extent possible. In this way, we smooth the color pattern and emphasize the real similarities and dissimilarities among sequences and blocks.

The *CIE LCH color space*, also known as polar-Lab color space, is a cylindrical transformation of the CIE Lab space so that the a and b axes are converted to a polar coordinate system. The radial coordinate C measures *chroma* and the angular

coordinate H measures *hue*. LCH space makes it easier to rotate colors, by changing their *hue* value. The CIE LCH colors can be converted from CIE Lab colors using an R library called *colorspace* [20, 21].

Hue values are circular, meaning 0 and 2π are identical. Given two angles p and q, the difference between them is given by

$$\text{diff}(p, q) = [(p - q + \pi) \bmod 2\pi] - \pi$$

This calculation returns the difference in the interval $[-\pi, \pi)$.

Given a set A of angles a_i, the average angle $\overline{A}$ is

$$\overline{A} = \arctan \frac{\sum_{i=1}^{n} \sin a_i}{\sum_{i=1}^{n} \cos a_i}$$

and the variance $\text{Var}(A)$ is

$$\text{Var}(A) = \frac{\sum_{i=1}^{n} \text{diff}(\overline{A}, a_i)^2}{n}$$

Given a colored MSA Z with m rows and n columns, to estimate its overall noise level, a penalty function $\text{pen}(Z)$ is defined by

$$\text{pen}(Z) = \sum_{j=1}^{m} \text{Var}(h_{1j}, h_{2j}, \cdots, h_{nj})$$

where h_{ij} is the hue value of the symbol at the ith column and jth row.

We look for an optimized set, R, of n rotation angles, r_i, such that after rotating each hue in every ith column of Z clockwise by r_i, the rotated MSA Z' achieves the minimized return value of the penalty function $\text{pen}(Z')$.

The optimization procedure is performed by R's general-purpose optimization function *optim()* [13]. Among five available optimization algorithms provided by this function, we choose the *bounded Broyden-Fletcher-Goldfarb-Shanno* (L-BFGS-B) method [22]. This algorithm allows box constraints, meaning each variable can be given a lower and/or upper bound. Plus, it uses a limited amount of memory. The initial values are set to 0 and boundaries to $[0, 2\pi]$. More detailed comparison between different optimization functions and algorithms will be discussed later in Chapter 6.

3.4 Hue Flipping

One characteristic of the coordinates generated from multidimensional scaling is that they can not only be rotated, but also flipped, without changing the distances between each other. Adding flipping ability to the above rotation optimization procedure may result in lower minimal penalty values and better coloring results. To keep the penalty function simple and fast, we perform a heuristic flipping *before* the rotation step.

For each pair of adjacent columns C_i and C_{i+1}, $1 \leq i \leq n-1$, compare their hues row by row. Rows with gaps in these two columns are not taken into consideration. For those rows with non-gap symbols on both sides, record their hue values

$$h_{i,1}, h_{i+1,1}, h_{i,2}, h_{i+1,2}, \cdots, h_{i,k}, h_{i+1,k}$$

into two lists H_i and H_{i+1}:

$$H_i = h_{i,1}, h_{i,2}, \cdots, h_{i,k}$$

$$H_{i+1} = h_{i+1,1}, h_{i+1,2}, \cdots, h_{i+1,k}$$

In both lists, compare two neighboring hue values $h_{i,j}$ and $h_{i,j+1}$, $1 \leq j \leq k-1$, using diff() function described in section 3.3. If $\text{diff}(h_{i,j}, h_{i,j+1}) < -\frac{\pi}{8}$, the change from $h_{i,j}$ to $h_{i,j+1}$ is called a *clockwise change*; if $\text{diff}(h_{i,j}, h_{i,j+1}) > \frac{\pi}{8}$, the change is called a *counterclockwise change*; otherwise, it's considered to be unchanged.

If a column contains more clockwise changes than counterclockwise changes, it is called *clockwise column*; if it contains more counterclockwise changes, it is called *counterclockwise column*; otherwise, it is a neutral column. Note that since gap rows are filtered out of the comparison, the type of a column may change when compared to different adjacent columns. For instance, a column may be clockwise compared to the previous column, and counterclockwise compared to the next column.

If two adjacent columns are in opposite types, flip the latter one to make them in same type. Then the flipped alignment are sent back to the procedure discussed in Section 3.3.

3.5 Sequence Sorting

The alignment graph is supposed to provide information on how sequences will group with each other. We sort sequences after coloring optimization, based on the average

hue of each one, which is calculated by the average angle function proposed previously in section 3.3.

The fact that hue values are circular, results in an additional question on where to cut the circle to get a sorted list of sequences. In a sorted circle of n average hue values, $h_1, h_2, \cdots, h_n$, we calculate the absolute difference between each pair of adjacent ones (h_1 is adjacent to h_n).

$$\Delta h_{1,2} = |\text{diff}(h_1, h_2)|$$

$$\Delta h_{2,3} = |\text{diff}(h_2, h_3)|$$

$$\cdots$$

$$\Delta h_{n,1} = |\text{diff}(h_n, h_1)|$$

Then, the circle is split at the biggest gap

$$\Delta h_{max} = \Delta h_{i,i+1} = |\text{diff}(h_i, h_{i+1})|$$

to best avoid separating similar sequences.

3.6 Time Complexity

Mavis has two time-consuming procedures in its MSA coloring algorithm. The first one is the multidimensional scaling which maps the distance matrix onto a high-dimensional space. The second one is the color optimization over rotation and flipping. For an MSA with m sequences and n columns, we analyze the time complexity of both procedures in this section.

Algorithm 1 shows the pseudocode of the multidimensional scaling procedure. This algorithm loops through n columns. For each column, the number of residues is no more than m, which is the number of sequences, so the size of the distance matrix is no more than $m \times m$. The time complexity of the classic multidimensional scaling function with an $m \times m$ distance matrix is $O(m^3)$ [23]. Therefore, the time complexity of our scaling procedure is $O(m^3 n)$.

Algorithm 1 Multidimensional Scaling Procedure

```
scores ← All similarity scores
n ← Number of columns
coords ← Empty set of coordinates
for i = 1 → n do
    scores[column] ← Subset of scores for the ith column
    distances ← Convert all scores[column] into distances
    matrix ← Distance matrix from distances
    coords[column] ← Classic multidimensional scaling on matrix (O(m^3))
    coords[column] ← Pick first two axes from coords[column]
end for
return coords
```

In order to optimize color patterns by rotation and flipping, we define the penalty function, whose pseudocode is shown in Algorithm 2, to assess the overall color noise level. Then we run L-BFGS-B algorithm on this penalty function to search for its the global minimum value.

The penalty function consists of two steps. First, it applies a certain amount of rotation and flipping to each residue in the alignment, which takes $O(mn)$ time. Second, it loops though all m rows, calculates the hue variance of each row, and sums them up. Since the calculation of hue variance on n residues takes $O(n)$, this step takes $O(mn)$ time in total. So the total time complexity of penalty function is

$O(mn)$.

The time complexity of L-BFGS-B algorithm is $O(mk)$ where k is a moderate number representing the limited amount of memory used in the algorithm [22]. Therefore, the time complexity of the color optimization procedure is $O(m^2nk)$. Since $k < m$, it can also be considered as $O(m^3n)$.

Algorithm 2 Coloring Penalty Function

```
for i = 1 → n do
  for j = 1 → m do
    Apply rotation and flipping to the color in the ith column and the jth row
  end for
end for
sumvar ← 0
for j = 1 → m do
  rowvar ← Hue variance of the jth row (O(n))
  sumvar ← sumvar + rowvar
end for
return sumvar
```

As a result, we estimate that the overall time complexity of Mavis algorithm is $O(m^3n)$.

Chapter 4

Implementation

We implemented Mavis' algorithm and user interface on a Mac OS X environment, while it is designed to work on other major platforms such as Microsoft Windows or Linux. The implementation consists of an R script as back-end to perform calculations and generate colors, a Perl CGI script as API, and an HTML and JavaScript web page as the user interface.

The color-generating script, *coloring.R*, is the core part of Mavis. It takes the pair-wise similarity score file, *color.tsv*, as input, performs multidimensional scaling and color optimization, sorts sequences, and output colors and sequence information into two tab-separated files, *colors.tsv* and *seqinfo.tsv*. The front-end web page takes an alignment ID as parameter and sends it to the API. The Perl CGI script, *api.pl*, parses the color file and sequence information file, along with the FASTA-format alignment file, and sends them back to front-end in JSON format [24]. The front-end uses a JavaScript, *mavis.js*, to generate the alignment matrix and render it with

colors and symbol letters.

The source code of *coloring.R*, *api.pl*, and *mavis.js* are available in the Appendix.

4.1 Color-generating Script

The script *coloring.R* begins by reading three arguments from command-line calls: score file name, colors file name, and sequence information file name. The first file is for input and the latter two are for output. An example of calling this script on UNIX operating system is given below.

```
Rscript coloring.R data/score.tsv data/colors.tsv data/seqinfo.tsv
```

The file *score.tsv* contains four columns separated by tabs: column number, first row number, second row number, and similarity score between these two rows in this column. The row and column numbers start from 1, and the similarity score values range from 0 to 1. Figure 4.1 shows a few example lines in a *score.tsv* file.

The script *coloring.R* consists of four parts. The first part reads the input file, performs multidimensional scaling and create color for each symbol in the CIE LCH space. The second part rotates and flips color hues and optimize the penalty function. The third part sorts the sequences by their average hues and calculates each sequences average RGB color. The last part outputs the symbol colors and sequence information into corresponding files.

#COL_NUMBER	#ROW_NUMBER_1	#ROW_NUMBER_2	#RES_PAIR_HIT/NALT
1	1	2	1.000000
1	1	6	0.933333
1	1	10	0.866667
1	2	6	0.933333
1	2	10	0.866667
1	6	10	0.133333
2	1	6	1.000000
2	1	10	1.000000
2	6	10	1.000000

Figure 4.1: An example of similarity score file *score.tsv*, showing the format of the file. The four columns are: column number, the first row number, the second row number, score value.

The color output file, *colors.tsv*, has five columns separated by tabs: column number, row number, and the symbol color in RGB triplet (red, green, blue) in the range from 0 to 1, which will be converted into hexadecimal format used in the user interface module. The sequence information file seqinfo.tsv also has five columns: row number, average hue from 0 to 360 degrees, and average color in RGB triplet. Figure 4.2 shows how a typical *color.tsv* file looks like.

4.2 API

A Perl CGI script, *api.pl*, and a Perl package *Mavis::API* act as the API between the R back-end and HTML/JavaScript front-end. The Perl CGI *api.pl* takes two parameters from HTTP requests, *action* and *id*. The default action is *alignment*, meaning the alignment information, including sequences, colors, average colors and order of sequences, should be returned. In this action, an additional *id* parameter is

```
1       1       0.1065524       0.5616907       0.3943046
1       2       0.106256        0.5616652       0.3936842
1       6       0.1065487       0.5616904       0.3942968
1       10      0.1065581       0.5616912       0.3943165
2       1       0.09057406      0.5677889       0.2867890
2       2       0.09057406      0.5677889       0.2867890
2       6       0.09057406      0.5677889       0.2867890
2       10      0.09057406      0.5677889       0.2867890
3       1       0.07634505      0.5664451       0.2581746
3       6       0.1113592       0.570106        0.3250781
3       10      0.07634505      0.5664451       0.2581746
```

Figure 4.2: An example of coloring result file *color.tsv*, showing the format of the file. The five columns are (from left to right): row number, average hue, red, green and blue.

expected as the alignment ID. Another available action is *list*, which requests a list of all existing alignment IDs.

The API returns the query result to the web page in JSON (JavaScript Object Notation) format, which is a lightweight text-based data-interchange standard. JSON is derived from a subset of JavaScript syntax and can be easily parsed in JavaScript as well as many other languages, like Perl. We included the Perl JSON 2.51 module to provide JSON encoding/decoding ability to our API [24].

Figure 4.3 shows an example of the API response in JSON format. The alignment ID is "demo". It contains two sequences, namely "Seq01" and "Seq02". Both sequences have the same content "DEMO". The status of this API response is 1, which means *ready*. For more details, please refer to the source code attached at the end of this report.

```
{
  "status"    : 1,
  "id"        : "demo",
  "colors"    : [
    [ "#666", "#666", "#666", "#666" ],
    [ "#665", "#667", "#667", "#665" ]
  ],
  "sequences" : [
    { "name": "Seq02", "color": "#26e", "seq": "DEMO" },
    { "name": "Seq01", "color": "#26e", "seq": "DEMO" }
  ]
}
```

Figure 4.3: An example of Mavis API response in JSON format.

4.3 User Interfaces

We implement a web interface with HTML and JavaScript to send API requests, process API responses, and draw the MSA matrix with colors.

The JavaScript code communicates with API using Ajax requests, which means that the request and response are performed asynchronously, without refreshing the web page. When JavaScript receives the Ajax response from the API, it updates the alignment matrix with colors on the web page. If the coloring data is not ready, the script periodically check with the API and updates the web page as soon as the data becomes available.

Figure 4.4 shows the web user interface of Mavis displayed in Firefox 5 and Mac OS X 10.6. The leftmost bar shows the average hue of each sequence. The sequences are sorted by this hue value, in order to help users group them. The sequence names

are displayed to the right of the average hue bar. Current date and time are added at the bottom-right corner. If Mavis has more than one colored alignment available, there will be a alignment list on the bottom of the page.

Users can choose to display sequence residue symbols or not by toggling a button at the top-left corner. Figure 4.5 shows the colored alignment without symbols displayed, which provides a clearer view of color patterns.

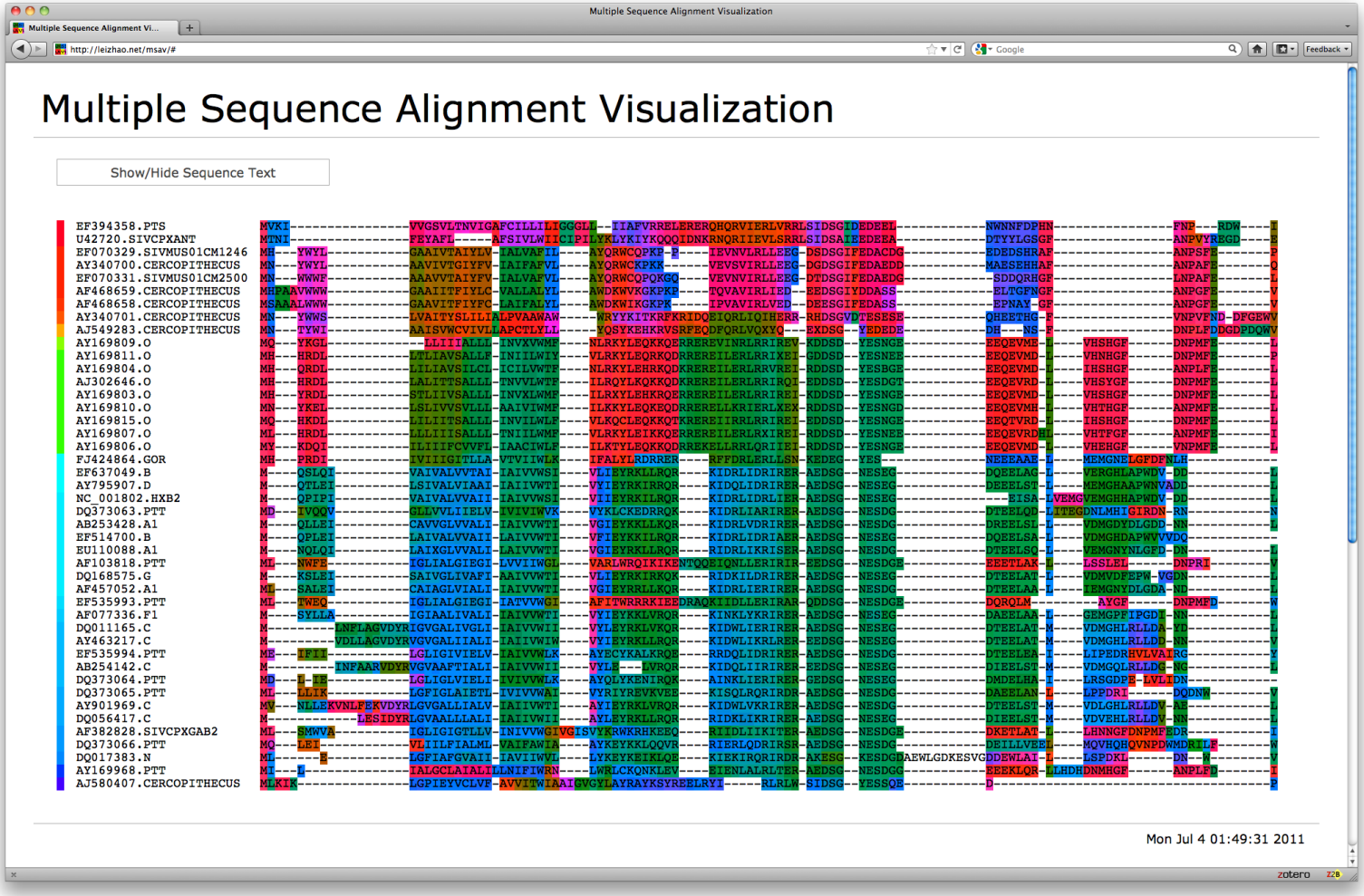


Figure 4.4: An MSA rendered by Mavis displayed in Firefox 5.

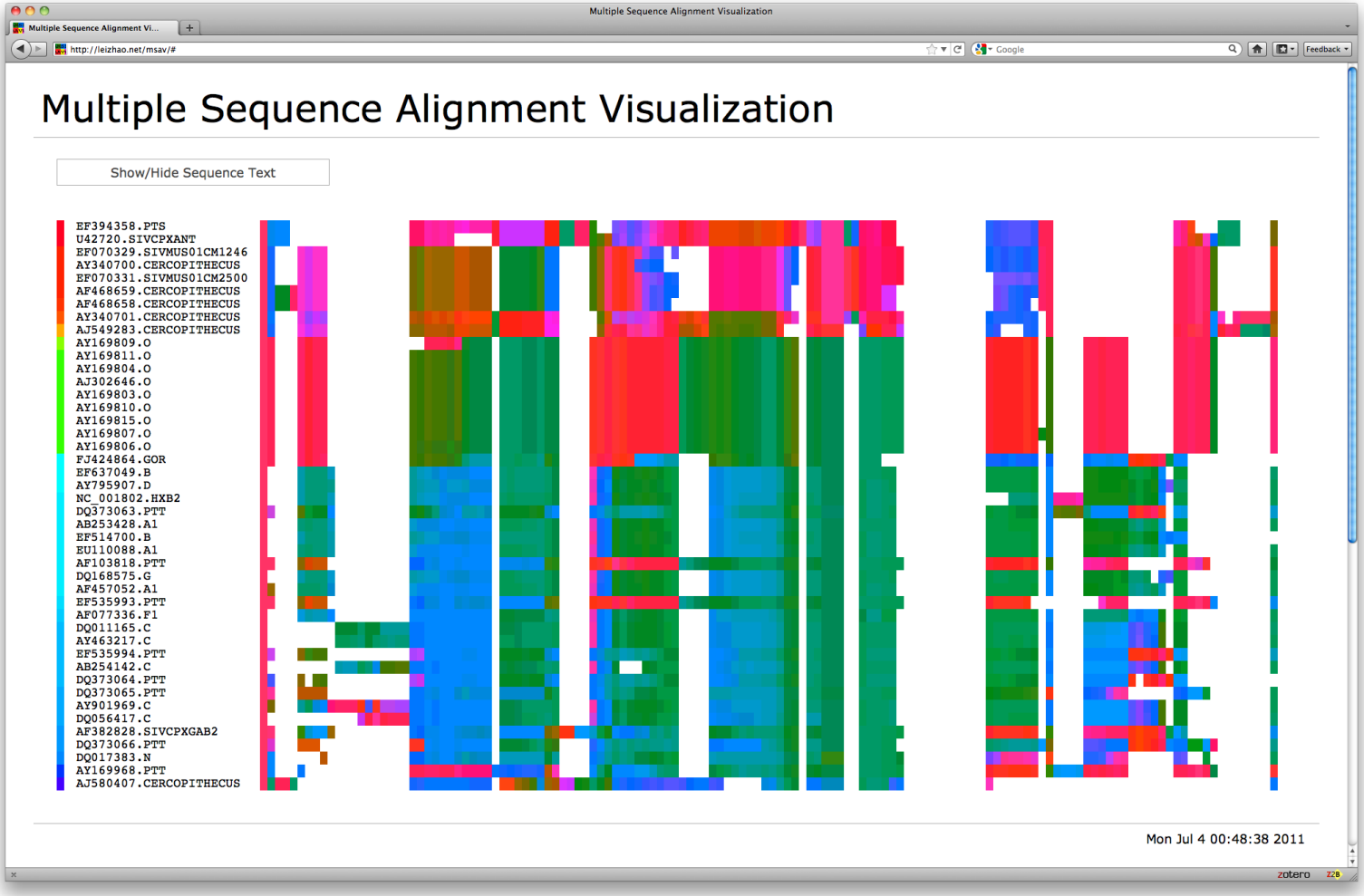


Figure 4.5: An MSA rendered by Mavis displayed in Firefox 5. Sequence residue symbols are hidden.

Chapter 5

Results and Tests

5.1 Validation

Mavis' primary goal is to represent alignment similarity by color similarity. In order to validate our algorithm, we generated three sets of artificial similarity data and applied Mavis on them. Each data set contains a small number of similarity scores with simple patterns and trends. For simplicity, all the residues have been designated as gaps ('-'), because residue types do not affect the result of Mavis. The effectiveness of the algorithm is easily revealed in the coloring results, by comparing the color patterns with the original data trends.

The first data set consists of two sequences of length 21. In the leftmost column, two residues have the highest similarity score, 1; in the rightmost column, the lowest similarity score, 0; in the other 19 columns in between, the scores decrease from left

to right in the step of 0.05 (see Table 5.1).

Table 5.1: Similarity Data Set for Validation Test #1

Columns	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	-	-	-	-	-	-	-	-	-	-	-
Sequence 2	-	-	-	-	-	-	-	-	-	-	-
Similarity Score	1	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6	0.55	0.5

Columns	12	13	14	15	16	17	18	19	20	21
Sequence 1	-	-	-	-	-	-	-	-	-	-
Sequence 2	-	-	-	-	-	-	-	-	-	-
Similarity Score	0.45	0.4	0.35	0.3	0.25	0.2	0.15	0.1	0.05	0

Figure 5.1 is the MSA matrix colored by Mavis. From left to right, the color difference between two sequences increases from very little to very significant in a gradual way. This result perfectly meets our expectation that the colors are not only representing the residue relationships within a column, but also rotated to become as smooth as possible within rows.

Figure 5.1: The coloring result of validation test #1.

The second data set is similar to the previous test, but consists of three sequences. Each column has three residues, of which any pair has the same similarity score. Similar to the first test, the scores decrease from 1 on the left to 0 on the right, in the step of 0.05 (see Table 5.2).

Figure 5.3 is the MSA matrix of the second test. Similarly, the color differences between any pair of the three sequences gradually increase from very little on the left

Table 5.2: Similarity Data Set for Validation Test #2

Columns	1	2	3	4	5	6	7	8	9	10	11
Sequence 1	-	-	-	-	-	-	-	-	-	-	-
Sequence 2	-	-	-	-	-	-	-	-	-	-	-
Sequence 3	-	-	-	-	-	-	-	-	-	-	-
Similarity Score $s_{1,2}$	1	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6	0.55	0.5
Similarity Score $s_{2,3}$	1	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6	0.55	0.5
Similarity Score $s_{1,3}$	1	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6	0.55	0.5

Columns	12	13	14	15	16	17	18	19	20	21
Sequence 1	-	-	-	-	-	-	-	-	-	-
Sequence 2	-	-	-	-	-	-	-	-	-	-
Sequence 3	-	-	-	-	-	-	-	-	-	-
Similarity Score $s_{1,2}$	0.45	0.4	0.35	0.3	0.25	0.2	0.15	0.1	0.05	0
Similarity Score $s_{2,3}$	0.45	0.4	0.35	0.3	0.25	0.2	0.15	0.1	0.05	0
Similarity Score $s_{1,3}$	0.45	0.4	0.35	0.3	0.25	0.2	0.15	0.1	0.05	0

to very significant on the right. In addition to the expectation of preserving residue distances and color rotation, this figure also illustrates how color flipping affects the result. For comparison, Figure 5.2 is the result from the same data set, but the flipping optimization is manually disabled. Obviously, colors in a few columns are not be able to match adjacent ones. The coordinates of these three residues are arranged in the two-dimensional color space as an isosceles triangle, in which the three vertices may be clockwise or counterclockwise. If both situations exist in this MSA matrix, their color patterns can not match, no matter how they are rotated.

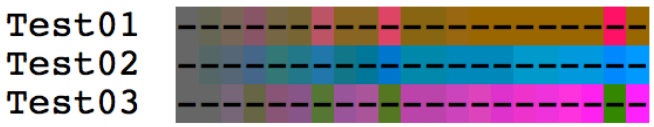


Figure 5.2: The coloring result of validation test #2 without color flipping.

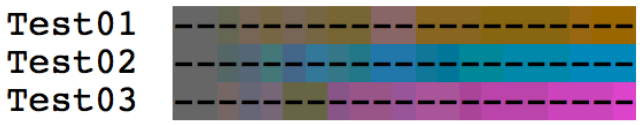


Figure 5.3: The coloring result of validation test #2 with color flipping.

The third data set consists of four sequences, which are divided into two groups. Within each column, any two residues from the same group have the maximum similarity score, meaning that they are exactly the same; any two residues from different groups have the same similarity score. Similarly, the scores decrease from 1 on the left to 0 on the right, in the same way as the first and second tests (see Table 5.3).

Table 5.3: Similarity Data Set for Validation Test #3

Columns	1	2	3	4	5	6	7	8	9	10	11
Sequence 1 (Group 1)	-	-	-	-	-	-	-	-	-	-	-
Sequence 2 (Group 1)	-	-	-	-	-	-	-	-	-	-	-
Sequence 3 (Group 2)	-	-	-	-	-	-	-	-	-	-	-
Sequence 4 (Group 2)	-	-	-	-	-	-	-	-	-	-	-
Similarity Scores $s_{1,2}$ and $s_{3,4}$	1	1	1	1	1	1	1	1	1	1	1
Similarity Scores $s_{1,3}$, $s_{1,4}$, $s_{2,3}$ and $s_{2,4}$	1	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6	0.55	0.5

Columns	12	13	14	15	16	17	18	19	20	21
Sequence 1 (Group 1)	-	-	-	-	-	-	-	-	-	-
Sequence 2 (Group 1)	-	-	-	-	-	-	-	-	-	-
Sequence 3 (Group 2)	-	-	-	-	-	-	-	-	-	-
Sequence 4 (Group 2)	-	-	-	-	-	-	-	-	-	-
Similarity Scores $s_{1,2}$ and $s_{3,4}$	1	1	1	1	1	1	1	1	1	1
Similarity Scores $s_{1,3}$, $s_{1,4}$, $s_{2,3}$ and $s_{2,4}$	0.45	0.4	0.35	0.3	0.25	0.2	0.15	0.1	0.05	0

The result of this data set is shown in Figure 5.4. First, each sequence has the correct gradient color pattern. Second, sequences from different groups are colored in different hues, reflecting the alignment distances between groups. Third, sequences from the same group share the same color pattern, reflecting the zero distances within a group. This result shows that Mavis is capable of handling more complex situations.

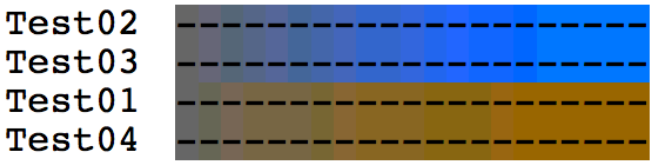


Figure 5.4: The coloring result of validation test #3.

Chapter 6

Discussion

6.1 Color Spaces

Since we are to convert the scaled coordinates to colors, choosing a suitable color space is an fundamental task. There are a number of available color spaces, some of them are listed in Table 6.1.

Table 6.1: Partial List of Color Spaces

Name	Description
RGB	additive color model based on red, green, and blue lights
HSL	a transformation of RGB, with hue, saturation, and lightness
CMYK	additive color model based on cyan, magenta, yellow, and black
CIE 1931 XYZ	the first color space based on measurements of human perception
CIE LUV	a modification of CIE 1931 XYZ
CIE Lab	more perceptually linear than other color spaces
CIE LCH	polar form of CIE Lab

RGB is one of the most commonly used color spaces. At first, we scaled the

distance matrix down to a three-dimensional space and mapped it directly to the RGB color space. However, we soon noticed that some colors, especially very dark ones and very light ones, did not perfectly serve the purpose of representing distances, yet made the graph more noisy. We also considered HSL and CMYK, but none of them reflect human eye perception well enough. The color space we need is a perceptually linear one, which means that a change of the same amount in a color value should represent a change of approximately the same visual importance.

After comparison and testing, we decided to choose CIE Lab color space, which best meets our needs. We also realized that using the whole color space and all possible colors is not necessary. Dark colors and light colors are not easy to be recognized and compared. Those with proper range of lightness are good enough for visualization purpose. So we decided to use only two dimensions of CIE Lab color space with a fixed lightness value (75). The algorithm reduces the higher dimensional space into a two-dimensional one and map it to this Lab space.

The reason why we didnt choose one-dimensional scaling is that a linear space either could not map to enough number of colors (for example, use only one primary color, like red), or could not preserve the distance information (for example, use only hues). We find two dimensions a good balance.

6.2 Optimization Algorithms

In R, there are several general purpose optimization packages which offer facilities for solving our color rotation and flipping problems. Two popular functions are optim()

and nlminb() from package stats.

Function optim() provides implementations of five algorithms: *Broyden-Fletcher-Goldfarb-Shanno (BFGS)*, *bounded BFGS (L-BFGS-B)*, *conjugate gradient (CG)*, *Nelder and Mead (Nelder-Mead)*, and *simulated annealing (SANN)*. *Nelder-Mead* [25], which is the default one, returns robust results but is relatively slow. *CG* [26] in faster in larger optimization problems, but more fragile. *BFGS* is a balance, and *L-BFGS-B* further provides the ability of box constraints, that each variable can be given a lower/upper bound. *SANN* [27] is more powerful on rough surfaces but relatively slow. Another function *nlminb()* offers similar box constraint optimization and similar performance to *L-BFGS-B*, so these two algorithms are chosen to a further test.

Optimization algorithms always suffer from the local versus global minimum problem, and the final result are more or less unstable and depending on the initial values. To decide which one of *L-BFGS-B* and *nlminb()* is more stable in our approach, we run a test on both of them. The test dataset is the alignment of 44 Vpu protein sequences from GUIDANCE server [9]. We randomly created 100 sets of initial values, performed optimizations, and see how the return value of the penalty function (described in section 3.3) changed. Table 6.2 shows the results of the comparison.

Table 6.2: Comparison of Optimization Functions *L-BFGS-B* and *nlminb()*

Algorithm	Min. Penalty	Avg. Penalty	Max. Penalty	Std. Deviation
Initial	370,584	425,728	444,771	13,998
L-BFGS-B	136,909	178,850	247,218	17,871
L-BFGS-B (3 times)	136,909	178,850	247,218	17,871
mlninb()	146,759	181,688	426,535	34,579
mlninb() (3 times)	142,888	171,380	325,469	21,402

Although running *mlninb()* three times results in the best and lowest average penalty value, it is not as good as *L-BFGS-B* in terms of best case score (minimum penalty value), worst case score (maximum penalty value), and standard deviation. In addition, *L-BFGS-B* reaches its best result in the first round, while *mnlinb()* needs more rounds and longer time to lower the penalty score to an acceptable level. Thus, we believe that *L-BFGS-B* is a more stable choice for our algorithm.

Chapter 7

Conclusion

In this report, we introduced Mavis, a new way of coloring and visualizing multiple sequence alignments. Our main objective has been to create an intuitive way of showing the quality and internal structure of an alignment. We use colors, which is the most natural and sensible way for human eyes, to accomplish this goal. The basic idea is, the greater the dissimilarity between sequences, the more obvious difference in colors.

Most previous techniques were based on fixed color schemes, meaning each sequence symbol, like an amino acid or a nucleotide, is designated to a pre-selected color. This is of course a simple and straightforward solution, and is easy for observers to find a particular type or pattern of symbols. However, this fixed-scheme approach does not emphasize the relationship between adjacent columns and the internal regions and structures in the level of the whole alignment. This is the main reason why we do not use any predetermined color schemes.

In order to dynamically generate colors for an MSA, we require a user-input data set of symbol-symbol similarity (or alignment confidence) scores. This data set is converted into a set of distance matrices, one for each alignment column. Each distance matrix is then scaled to a two-dimensional space and mapped onto a CIE Lab color space to get colors. To optimize the result, the color space is rotated and flipped to reduce the overall color noise level as low as possible. This strategy brings significant improvement to the alignment coloring, by showing the well aligned regions in solid colors, while poorly aligned ones in various different colors.

Mavis is implemented in R, Perl, HTML, and JavaScript. The core algorithm written in R performs all the calculations. Perl CGI is used to set up an API between back-end and front-end. The API is designed in an extensible way and supports different user interfaces, even user-created ones. We provide a graphical user interface in HTML and JavaScript, which can be easily accessed over the Internet with any major web browsers.

We hope Mavis becomes a fast, light-weight, and easy-to-use tool for biologists who study comparative sequence data, and assists researchers to better analyze multiple sequence alignments, especially from functional, structural, and evolutionary perspectives.

References

[1] A. Cornish-Bowden, "Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984.," *Nucleic Acids Research*, vol. 13, pp. 3021–3030, May 1985. PMID: 2582368 PMCID: 341218.

[2] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler, "GenBank," *Nucleic Acids Research*, vol. 36, pp. D25–D30, Dec. 2007.

[3] R. C. Edgar and S. Batzoglou, "Multiple sequence alignment," *Current Opinion in Structural Biology*, vol. 16, no. 3, pp. 368–373, 2006.

[4] J. D. Thompson, T. J. Gibson, and D. G. Higgins, "Multiple sequence alignment using ClustalW and ClustalX.," *Current Protocols in Bioinformatics*, p. 2.3, 2002.

[5] A. M. Waterhouse, J. B. Procter, D. M. A. Martin, M. Clamp, and G. J. Barton, "Jalview version 2—a multiple sequence alignment editor and analysis workbench," *Bioinformatics*, vol. 25, pp. 1189 –1191, May 2009.

[6] J. Procter, J. Thompson, I. Letunic, C. Creevey, F. Jossinet, and G. Barton, "Visualization of multiple alignments, phylogenies and gene family evolution," *Nature Methods*, vol. 7, no. 3 Suppl, pp. S16–25, 2010.

[7] K. Lin, A. C. W. May, and W. R. Taylor, "Amino acid encoding schemes from protein structure alignments: multi-dimensional vectors to describe residue types," *Journal of Theoretical Biology*, vol. 216, no. 3, pp. 361 – 365, 2002.

[8] O. Penn, E. Privman, G. Landan, D. Graur, and T. Pupko, "An alignment confidence score capturing robustness to guide tree uncertainty," *Molecular Biology and Evolution*, vol. 27, no. 8, pp. 1759 –1767, 2010.

[9] O. Penn, E. Privman, H. Ashkenazy, G. Landan, D. Graur, and T. Pupko, "GUIDANCE: a web server for assessing alignment confidence scores," *Nucleic Acids Research*, vol. 38, no. Web Server, pp. W23–W28, 2010.

[10] I. Borg and P. J. F. Groenen, *Modern multidimensional scaling: theory and applications*. Springer, 1997.

[11] K. McLaren, "XIII—The development of the CIE 1976 (L* a* b*) uniform colour space and colourdifference formula," *Journal of the Society of Dyers and Colourists*, vol. 92, no. 9, pp. 338–341, 1976.

[12] J. C. Gower, "Some distance properties of latent root and vector methods used in multivariate analysis," *Biometrika*, vol. 53, no. 3-4, pp. 325–338, 1966.

[13] R Development Core Team, *R: a language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[14] R. Gentleman and R. Ihaka, "R: a language for data analysis and graphics," *Journal Of Computational And Graphical Statistics*, vol. 5, no. 3, pp. 299–314, 1996.

[15] F. Cailliez, "The analytical solution of the additive constant problem," *Psychometrika*, vol. 48, no. 2, pp. 305–308, 1983.

[16] M. A. A. Cox and T. F. Cox, "Multidimensional scaling," in *Handbook of Data Visualization*, pp. 315–347, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[17] D. Margulis, *Photoshop LAB color: The canyon conundrum and other adventures in the most powerful colorspace*. Peachpit Press Berkeley, CA, USA, 2005.

[18] C.I.E., *Commission internationale de l'Eclairage proceedings, 1931*. Cambridge University Press, Cambridge, 1932.

[19] T. Smith, "The C.I.E. colorimetric standards and their use," *Transactions of the Optical Society*, vol. 33, no. 3, pp. 73–134, 1931.

[20] R. Ihaka, P. Murrell, K. Hornik, and A. Zeileis, *colorspace: color space manipulation*, 2009. R package version 1.0-1.

[21] A. Zeileis, K. Hornik, and P. Murrell, "Escaping RGBland: selecting colors for statistical graphics," *Computational Statistics & Data Analysis*, vol. 53, pp. 3259–3270, 2009.

[22] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, p. 1190, 1995.

[23] J. Tzeng, H. Lu, and W. Li, "Multidimensional scaling for large genomic data sets," *BMC Bioinformatics*, vol. 9, no. 1, p. 179, 2008.

[24] D. Crockford, "The application/json media type for javascript object notation (json)," tech. rep., RFC 4627, July, 2006.

[25] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308 –313, 1965.

[26] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The Computer Journal*, vol. 7, no. 2, pp. 149 –154, 1964.

[27] C. J. P. Bélisle, "Convergence theorems for a class of simulated annealing algorithms on rd," *Journal of Applied Probability*, vol. 29, no. 4, pp. 885–895, 1992.

Appendix A

Color-generating Script

coloring.R

```
1  library("colorspace")

3  args <- commandArgs(TRUE)

5  FILE.SCORE <- args[1]
   FILE.COLOR <- args[2]
7  FILE.ORDER <- args[3]

9  scaleScore <- function(filename) {
     rawData  <- read.table(filename)
11   nCols    <- max(rawData[, 1])
     nRows    <- max(rawData[, 2], rawData[, 3])
13   scaled   <- data.frame()
```

```
15      for (col in 1 : nCols) {
          colData  <- rawData[rawData[, 1] == col, ]
17        if (length(colData[, 1]) == 0) next
          colData[, 4] <- 1 - colData[, 4]
19        points   <- sort(unique(c(colData[, 2], colData[, 3])))
          nPoints  <- length(points)
21
          distMat  <- matrix(0, nPoints, nPoints)
23        dimnames(distMat) <- list(points, points)
          distInd  <- rbind(cbind(match(colData[[2]], points), match(
              colData[[3]], points)),
25                          cbind(match(colData[[3]], points), match(
                                colData[[2]], points)))
          distMat[distInd]  <- rep(colData[[4]], 2)
27
          colScaled <- cmdscale(distMat, min(nrow(distMat) - 1, 2))
29        colScaled <- cbind(col, points, colScaled, if (ncol(colScaled) <
              2) 0)
          if (ncol(colScaled) < 4) colScaled <- cbind(colScaled, 0)
31        colScaled[is.nan(colScaled[, 4]), 4] <- 0
          scaled    <- rbind(scaled, colScaled)
33      }
        scaled
35    }

37    createLch    <- function(scaled) {
        lightness  <- 70
```

```
39      multiplier <- 100
        labData    <- cbind(lightness, scaled[, 3:4] * multiplier)
41      labColor   <- LAB(data.matrix(labData))
        lchColor   <- as(labColor, "polarLAB")
43      lchColor
    }
45
    adjustColor <- function(lchColor) {
47      lchData   <- coords(lchColor)
        chroma    <- lchData[, 2]
49      lightness <- (chroma / max(chroma)) * 0 + 70
        chroma    <- (chroma / max(chroma)) * 30 + 70
51      lchData   <- cbind(lightness, chroma, lchData[, 3])
        lchColor  <- polarLAB(lchData)
53      lchColor
    }
55
    colorToData <- function(color) {
57      cbind(scaled[, 1:2], coords(color))
    }
59
    angleDiff <- function(angleA, angleB) {
61      (angleA - angleB + 180) %% 360 - 180
    }
63
    angleAvg <- function(angles) {
65      sumSin <- sum(sin(angles / 180 * pi))
        sumCos <- sum(cos(angles / 180 * pi))
```

```
67      if (sumSin * sumCos == 0) 0 else (atan2(sumSin, sumCos) / pi *
            180)
    }
69
    angleVar  <- function(angles) {
71    if (length(angles) == 0) return(0)
      average <- angleAvg(angles)
73    sum(((average - angles + 180) %% 360 - 180) ^ 2) / length(angles)
    }
75
    angleAdd  <- function(angleA, angleB) {
77    (angleA + angleB) %% 360
    }
79
    doFlipping <- function(lchData) {
81    nCols    <- max(lchData[, 1])
      nRows    <- max(lchData[, 2])
83    for (col in 1 : (nCols - 1)) {
        hues   <- matrix(ncol = 2, nrow = 0)
85      colOne <- lchData[lchData[, 1] == col, ]
        colTwo <- lchData[lchData[, 1] == col + 1, ]
87      for (row in 1 : nRows) {
          hue1 <- colOne[colOne[, 2] == row, 5]
89        hue2 <- colTwo[colTwo[, 2] == row, 5]
          if (length(hue1) & length(hue2)) {
91          hues <- rbind(hues, c(hue1, hue2))
          }
93      }
```

```
    if (nrow(hues) < 2) next
95      changes <- hues[c(2 : nrow(hues), 1), ] - hues
      changes <- ifelse(abs(angleDiff(changes, 0)) < 15 | abs(
         angleDiff(changes, 180)) < 15, 0, changes)
97      direction <- ifelse(changes > 0, 1, 0)
      direction <- ifelse(changes < 0, -1, direction)
99      if (sum(direction[, 1]) * sum(direction[, 2]) < 0) {
        lchData[lchData[, 1] == col + 1, 5] = 360 - lchData[lchData[,
           1] == col + 1, 5]
101     }
    }
103   lchData
  }
105
  doRotation <- function(lchData, rotation) {
107   rotData  <- lchData
    rotData[, 5] <- angleAdd(rotData[, 5], rotation[rotData[, 1]])
109   rotData
  }
111
  penalty    <- function(rotation) {
113   rotData  <- doRotation(lchData, rotation)
    nRows    <- max(lchData[, 2])
115   sumVar   <- 0
    for (row in 1 : nRows) {
117     rowVar <- angleVar(rotData[rotData[, 2] == row, 5])
      sumVar <- sumVar + rowVar
119   }
```

```
    sumVar
  }

  initRot    <- function(lchData) {
    nCols    <- max(lchData[, 1])
    rotation <- rep(0, nCols)
    rotation
  }

  bfgsOptim  <- function(lchData, rotation) {
    nCols    <- length(rotation)
    optimRot <- optim(rotation, penalty, NULL, method = "L-BFGS-B",
       lower = rep(0, nCols), upper = rep(360, nCols))
    optimRot$par
  }

  rotSeqOrd  <- function(seqOrder) {
    seqHues  <- seqOrder[, 2]
    nextHues <- c(seqHues[-1], seqHues[1])
    diffHues <- abs(angleDiff(nextHues, seqHues))
    maxGap   <- which(diffHues == max(diffHues))[1]
    newOrder <- seqOrder[1 : maxGap, ]
    if (maxGap < nrow(seqOrder)) {
      newOrder <- rbind(seqOrder[(maxGap + 1) : nrow(seqOrder), ],
          newOrder)
    }
    newOrder
  }
```

```
147   sortSeq    <- function(rotData) {
        nRows    <- max(rotData[, 2])
149     sortData <- data.frame()
        for (row in 1 : nRows) {
151       rowData  <- rotData[rotData[, 2] == row, ]
          rowAvg   <- angleAvg(rowData[, 5]) %% 360
153       sortData <- rbind(sortData, c(row, rowAvg))
        }
155     seqOrder <- sortData[order(sortData[, 2]), ]
        seqOrder <- rotSeqOrd(seqOrder)
157     seqOrder
      }
159
      outputData <- function(rotData, seqOrder) {
161     lchColor <- polarLAB(data.matrix(rotData[, 3:5]))
        rgbColor <- as(lchColor, "RGB")
163     rgbData  <- cbind(rotData[, 1:2], coords(rgbColor))
        write(t(rgbData),  file = FILE.COLOR, sep = "\t", ncolumns = 5)
165     write(t(seqOrder), file = FILE.ORDER, sep = "\t", ncolumns = 2)
      }
167
      scaled   <- scaleScore(FILE.SCORE)
169   lchColor <- createLch(scaled)
      lchColor <- adjustColor(lchColor)
171   lchData  <- colorToData(lchColor)

173   lchData  <- doFlipping(lchData)
```

```
175     rotation <- initRot(lchData)
        rotation <- bfgsOptim(lchData, rotation)
177     rotData  <- doRotation(lchData, rotation)

179     seqOrder <- sortSeq(rotData)

181     outputData(rotData, seqOrder)
```

Appendix B

API Script

api.pl

```
1  #!/usr/bin/env perl

3  use warnings;
   use strict;
5  use CGI qw(:standard);
   use JSON;
7  use Mavis::API;

9  my $action   = param('action');
   my $align_id = param('id');
11
   print header('application/json');
13
```

```
if (!defined $action or $action eq 'alignment') {
  eval {
    print encode_json &load_alignment($align_id);
  }
  or do {
    print encode_json({ status => -1 });
  }
}
elsif ($action eq 'list') {
  opendir DATADIR, 'data' or die "Can't open data directory.";
  my @align_list = grep { /^[^\.]/ } readdir(DATADIR) or die "Can't
     read data directory.";
  closedir DATADIR;
  print encode_json [ @align_list ];
}
else {
  print encode_json({ status => -1 });
}
```

Mavis/API.pm

```
package Mavis::API;

use warnings;
use strict;
use vars qw($VERSION @ISA @EXPORT @EXPORT_OK);
require Exporter;
require AutoLoader;
```

```
8
    @ISA     = qw(Exporter AutoLoader);
10  @EXPORT  = qw(load_alignment);
    $VERSION = '2.0';
12
    sub alignment_file_path {
14    my $align_id = shift || 'demo';
      return "data/$align_id/alignment.fasta";
16  }

18  sub colors_file_path {
      my $align_id = shift || 'demo';
20    return "data/$align_id/colors.tsv";
    }
22
    sub order_file_path {
24    my $align_id = shift || 'demo';
      return "data/$align_id/order.tsv";
26  }

28  sub load_alignment {
      my $align_id  = shift;
30    my $sequences = &load_sequences($align_id);
      my $colors    = &load_colors($align_id);
32    my $alignment = { id => $align_id, status => 1, sequences =>
         $sequences, colors => $colors };
      return $alignment;
34  }
```

```
36  sub load_sequences {
      my $align_id = shift;
38    my $sequences;
      my $align_data = &load_alignment_file(&alignment_file_path(
         $align_id));
40    my $order_data = &load_sequence_order(&order_file_path($align_id))
         ;
      for my $order_info (@$order_data) {
42      my $seq_info = $align_data->[$order_info->{row} - 1];
        push @$sequences, {(%$seq_info, %$order_info)};
44    }
      return $sequences;
46  }

48  sub load_alignment_file {
      my $align_file = shift;
50    my $align_data;
      open ALIGN, $align_file or die "Input file ($align_file) is
         missing.\n";
52    our $/ = ">";
      for my $align_input (<ALIGN>) {
54      chomp $align_input;
        $align_input =~ /^(.*?)\n(.*)/s or next;
56      my ($name, $seq) = ($1, $2);
        $seq =~ s/\n//sg;
58      push @$align_data, { name => $name, seq => $seq };
      }
```

```
60      $/ = "\n";
        close ALIGN;
62      return $align_data;
    }
64
    sub load_sequence_order {
66      my $seq_order_file = shift;
        my $seq_order;
68      open ORDER, $seq_order_file or die "Input file ($seq_order_file)
            is missing.\n";
        for my $seq_info (<ORDER>) {
70        chomp $seq_info;
          my ($row, $seq_hue, $r, $g, $b, $l, $a_val, $b_val) = split /\t
              /, $seq_info;
72        push @$seq_order, { row => $row, color => &hue_to_html($seq_hue)
              , a => $a_val, b => $b_val };
        }
74      close ORDER;
        return $seq_order;
76  }

78  sub load_colors {
        my $align_id   = shift;
80      my $color_file = &colors_file_path($align_id);
        my $colors;
82      open COLORS, $color_file or die "Input file ($color_file) is
            missing.\n";
        for my $line (<COLORS>) {
```

```
84        chomp $line;
          next if $line =~ /^#/;
86        next unless $line =~ /^\s*(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+(\S+)$
             /;
          $colors->[$1][$2] = &rgb_to_html($3, $4, $5);
88      }
        close COLORS;
90      return $colors;
      }
92
      sub rgb_to_html {
94      my ($r, $g, $b) = @_;
        $r = 0 if $r < 0; $g = 0 if $g < 0; $b = 0 if $b < 0;
96      $r = 1 if $r > 1; $g = 1 if $g > 1; $b = 1 if $b > 1;
        return sprintf "#%01x%01x%01x", int($r * 15.99), int($g * 15.99),
           int($b * 15.99);
98    }

100   sub hue_to_html {
        my $hue = shift;
102     my $r   = &hue2prim($hue + 120);
        my $g   = &hue2prim($hue);
104     my $b   = &hue2prim($hue - 120);
        return &rgb_to_html($r, $g, $b);
106
        sub hue2prim {
108       my $h = (shift) % 360;
          if ($h < 60)  { return $h / 60; }
```

```
110         if ($h < 180) { return 1; }
            if ($h < 240) { return (240 - $h) / 60; }
112         return 0;
          }
114     }


116     1;
        __END__
```

Appendix C

User Interface Script

mavis.js

```
var alignmentID   = urlParam("id");
var showText      = true;
var DataStatus    = { "Ready" : 1, "Pending" : 0, "Error" : 2 };
var alignmentData = { "id" : alignmentID, "status" : DataStatus.
   Pending, "sequences" : [], "colors" : [] };

$(document).ready(function() {
  getAlignmentData(alignmentID);
  drawAlignmentLayout();
  drawAlignmentTable();
  drawAlignmentList();
});
```

```
13  function apiUrl(alignmentID, action) {
      var URL = "api.pl?id=" + alignmentID;
15    if (action) URL = URL + "&action=" + action;
      return URL;
17  }

19  function getAlignmentData(alignmentID) {
      $.getJSON(apiUrl(alignmentID), function(alignment) { alignmentData
          = alignment; refreshDisplay(); });
21  }

23  function refreshDisplay() {
      drawAlignmentLayout();
25    drawAlignmentTable();
      drawAlignmentColors();
27  }

29  function drawAlignmentLayout(title) {
      if (!title || title.length < 1)
31      title = "Mavis" + (alignmentID.length > 0 ? " (ID: " +
            alignmentID + ")" : "");
      $("#alignment_container").html("<div id='alignment_header'>" +
          title + "</div>");
33    $("#alignment_container").append("<a href='#' class='button'
          onclick='toggleAlignmentText()'>Show/Hide Sequence Text</a>");
      $("#alignment_container").append("<table id='alignment_table'
          cellspacing='0'></table>");
```

```
35      $("#alignment_container").append("<div id='alignment_footer'>" +
            localtime() + "</div>");
    }
37
    function drawAlignmentTable() {
39      if (alignmentData.status == DataStatus.Ready) {
          $("#alignment_table").empty();
41        $.each(alignmentData.sequences, function(i, sequence) {
            var rowHtml = "<tr><td id='seq_color_" + sequence.row + "'>&
                nbsp;</td><td class='seq_name'>" + sequence.name + "</td>";
43          $.each(sequence.seq.split(''), function(j, character) {
              var column = j + 1;
45            if (!showText) character = " ";
              rowHtml += "<td id='cell_" + sequence.row + "_" + column +
                  "'>" + character + "</td>";
47          });
            rowHtml += "</tr>";
49          $("#alignment_table").append(rowHtml);
          });
51      }
        else if (alignmentData.status == DataStatus.Pending) {
53        $("#alignment_table").html("<tr><td>Loading...</td></tr>");
          setTimeout("getAlignmentData(alignmentID)", 3000);
55      }
        else {
57        $("#alignment_table").html("<tr><td>Error!</td></tr>");
        }
59  }
```

```
61  function drawAlignmentColors() {
      $.each(alignmentData.colors, function(column, columnColors) {
63      if (columnColors != null) {
          $.each(columnColors, function(row, cellColor) {
65          $("#cell_" + row + "_" + column).css("background-color",
                cellColor);
          });
67      }
      });
69    $.each(alignmentData.sequences, function(i, sequence) {
      $("#seq_color_" + sequence.row).css("background-color", sequence
          .color);
71    });
  }
73
  function drawAlignmentList() {
75    $("#alignment_list").html("<div>Other Alignments:</div>");
      $.getJSON(apiUrl(alignmentID, "list"), function(list) {
77      $.each(list, function(i, aID) {
          $("#alignment_list").append("<li><a href='?id=" + aID + "'>" +
              aID + "</a></li>");
79      });
      });
81  }

83  function toggleAlignmentText() {
      showText = !showText;
```

```
85      refreshDisplay();
    }
87
    function urlParam(name) {
89      name = name.replace(/[\[]/,"\\\[").replace(/[\]]/,"\\\]");
        var regex   = new RegExp("[\\?&]" + name + "=([^&#]*)");
91      var results = regex.exec(window.location.href);
        return results ? results[1] : "";
93  }

95  function localtime() {
        return new Date().toLocaleString();
97  }
```