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

May 2011

MAVIS: VISUALIZATION TOOL FOR MULTIPLE SEQUENCE ALIGNMENTS

Lei Zhao

APPROVED:

Acknowledgements

(Under construction)

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

May 2011

Abstract

(Under construction)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Multiple sequence alignment (MSA) analysis, a fundamental approach in molecular biology, is required for nearly all comparative sequence studies, including evolutionary, functional and structural aspects. A multiple sequence alignment is usually a matrix, in which each row represents a sequence, and each column corresponds to the equivalent positions across all sequences aligned. Each element of the matrix contains a symbol, which can be either '-' (a gap) or a sequence symbol (an amino acid for DNA/RNA sequences, or a nucleotide base for protein sequences). [1]

Due to the size and complexity of the multiple sequence alignment datasets, computer visualization techniques are essential to assist in understanding, analyzing and evaluating the alignment results. 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 a grid of symbols, usually along with various coloring, sizing and annotation. [2]

1.2 Related Work

Coloring an alignment has been widely implemented to highlight specific regions, properties or patterns. [2] 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. There are numerous color schemes available for different purposes, but Taylor [3] and Clustal [4] are widely used and often considered de facto standards.

However, using pre-determined color schemes to color sequence symbols, either amino acids or nucleotide bases, are not always helpful to phylogenetic analysis. Those fixed schemes focus on chemical differences between specific types of symbols, rather than between sequences, regions, blocks or structures of the alignment. For example, if symbols are examined within a certain column, the colors reflect the similarities fairly well. However, if the alignment is analyzed horizontally, comparative information within a row or a rectangular region are unable to be displayed, like how one sequence is similar with others, which section aligns better than another, which subset of sequences in which range of columns are more conserved, or how robust is the alignment accuracy.

Here we suggest a new MSA visualization tool, Mavis (Multiple Alignment VISualization), which focuses on highlighting evolutionary relationships and internal structures of an alignment. Instead of relying on chemical properties or any other information from symbol types, our approach takes only symbol-symbol correlation into consideration. The most valuable result we would like to provide is a whole picture of the MSA.

1.3 Thesis Outline

This thesis is organized in the following way: Chapter 2 briefly summarizes how Mavis processes data input, creates colors and displays the colored MSA graph. Chapter 3 explains the algorithm step by step, from scaling to mapping and optimization. Chapter 4 describes our implementation of the algorithm, including the back-end, API and front-end. Chapter 5 presents test results to show the validity and performance of our implementation. The whole work is discussed and concluded in Chapter 6.

Chapter 2

Approach

In this thesis, we address a different method to visualize multiple sequence alignments. The main objective is to correlate alignment similarity (or confidence) with color similarity. In the alignment matrix, a well-aligned area forms a solid-color shape, while a poor-aligned one becomes a mosaic of various colors. Alignment-level information, such as evolutionary relationships, phylogenetic characteristics and alignment accuracy should be revealed by examining the graph generated by our tool.

2.1 Data Input

Unlike most traditional coloring approaches, we start with an set of pairwise scores, rather than the alignment itself. Each of these scores represents the similarity or alignment confidence between two symbols aligned together in the same position, or, in the same column. Every pair of symbols in the same column has a score.

This set of data can be generated by GUIDANCE [5, 6], a measure quantifying alignment uncertainty. It produces a confidence score between 0 and 1, named GUIDANCE score, for each symbol, symbol pair, column and sequence in the alignment. The symbol-pair GUIDANCE confidence scores can be used in our study to visualize MSAs.

2.2 Color Generating

The colors of the symbols are generated in such way that the color differences represent symbol differences. We consider the confidence scores reversely, as distances, and convert all the scores of a column into a $N * N$ distance matrix, where N is the number of symbols. These N symbols can be represented by N points in at most $N-1$ dimensions, such that the Euclidean distance between any pair of points is exactly equal to the distance in the matrix. Once we can map these points to a color space, those corresponding colors should be able to fulfill our requirements.

However, most color spaces are three-dimensional or under. It is necessary to reduce the dimension from $N-1$ to at most 3 in order to process the mapping, but still preserve the distances as much as possible. This dimension reduction is done by a statistical technique called *classical multidimensional scaling* [7].

2.3 Color Optimization

Up to now, our approach only operates on the column level. Further adjustment is still needed to achieve the alignment-level objectives. We notice that the color space can be arbitrarily rotated and/or flipped, without changing the distance between any pair of symbols. 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.

We defined a penalty function to quantify the overall noise level of an alignment. The ideal combination of rotating and flipping each column should result in the lowest penalty value. For large MSAs, finding the global optimum is not always possible, and the approximate solution is also an acceptable option.

2.4 MSA Visualization

We choose to implement a web-based user interface, because of its flexibility, portability and light weight. Many features are provided to enhance the visualization effects, such as sorting sequences by their average colors, toggling to show/hide symbol names, and multiple-MSA support. An API between the front-end and the back-end is designed to make the system more extensible.

Chapter 3

Algorithms

The algorithm of color computation consists of five stages. First, multidimensional scaling reduces the number of dimensions and convert the confidence/similarity scores into coordinates. Second, coordinates are mapped onto part of the *CIE Lab color space* [8], and colors are created. Third, colors are flipped and rotated to smooth the graph and remove color noises. Finally, calculate the average color/hue of each sequence, sort them, and output for further display.

3.1 Multidimensional Scaling

This stage begins with the confidence scores described above. Each column in the MSA has a complete set C of pairwise scores. Each pair of non-gap symbols i, j in this column has a score $c_{i,j} = c_{j,i} \in [0, 1]$. 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

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

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 non-gap symbols in the current column.

These n symbols are placed into a two-dimensional space, in which the distances between each other are still preserved as much as possible. This procedure involves a statistical technique called *classical multidimensional scaling (classical MDS)*, also known as *(*Torgerson-Gower scaling) or *principle coordinates analysis (PCoA)* [9]. Multidimensional scaling takes a distance matrix and assigns a coordinate to each item in a lower 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()* [10] is utilized in this multidimensional scaling procedure. This function is written in R, which is a programming language for statistical computing [11], 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. [12, 13]

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 [8] is a color-opponent space. 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 [14, 15]. 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. [16]

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, they introduced unnecessary confusion by using a fixed color scheme. Even for exactly same aligned columns, colors may be totally different, which won't 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 colors 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 axles 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* [17, 18].

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

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

This calculation returns the difference in the interval $[-\pi, \pi)$, or in degrees $[-180°, 180°)$. The absolute value of the difference

$$|\mathrm{diff}(p, q)| = |(p - q + \pi) \bmod 2\pi - \pi|$$

returns in the interval $[0, \pi]$, or in degrees $[0°, 180°]$.

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 $\mathrm{Var}(A)$ is

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

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

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

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()* [10]. Among five available optimization algorithms provided by this function, we choose the *bounded Broyden-Fletcher-Goldfarb-Shanno* (L-BFGS-B) method [19]. 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, 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 take 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 $\mathrm{diff}(h_{i,j}, h_{i,j+1}) < -\frac{\pi}{8}$, the change from $h_{i,j}$ to $h_{i,j+1}$ is called a *clockwise change*; if $\mathrm{diff}(h_{i,j}, h_{i,j+1}) > \frac{\pi}{8}$, 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's called *clockwise column*; if it contains more counterclockwise changes, it's called *counterclockwise column*; otherwise, it's 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.

Chapter 4

Implementation

We implemented the entire visualization pipeline on a Mac OS X server. The implementation consists of an R script as back-end, a Perl CGI script as API, and an HTML/JavaScript web page as user interface.

The R script *coloring.r* takes the pair-wise similarity score file 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 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 [20]. The front-end uses JavaScript to generate the alignment matrix and render it with colors and symbol letters.

4.1 R Script

The script *coloring.r* begins with 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.

```
and
```

Rscript coloring.r data/score.tsv data/colors.tsv data/seqinfo.tsv

The file score.tsv contains four columns separated by tabs: column number, row number one, row number two, and similarity score between these two rows in this column. The row and column numbers start from one, and the similarity score values range from zero to one.

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.

The color 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 HTML and JavaScript. The sequence information file seqinfo.tsv also has five columns: row number, average hue from 0 to 360 degrees, average color in RGB triplet.

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. In this action, an additional id parameter must be provided as the alignment ID. Another available action is list, which requests 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.

4.3 User Interface

We implement an web interface with HTML and JavaScript to send API requests and draw visualization graphs. The JavaScript code dynamically communicates with API, creates an HTML table, and fills in colors and symbols. This part is simplified by jQuery, which is a popular cross-platform JavaScript library that abstracts complicated DOM selection, CSS manipulation and Ajax effects.

Chapter 5

Results

To evaluate the validity and performance of our approach, we tested it in two different ways. First, we run it on a few artificial datasets, to see if it creates the color patterns as we expected. Then, we run it on the BAliBASE dataset, observe how alignment size, in both number of rows and columns, will affect the excution time, and compare it with other tools.

5.1 Validity Test

(Under construction)

5.2 Performance Test

(Under construction)

Chapter 6

Discussion

In this report, we introduced a new way of coloring and visualizing multiple sequence alignments based on symbol-symbol similarity scores. Our main objective has been to create an intuitive way of showing the quality and internal structure of an alignment, using colors, which is the most natural and sensible way for human eyes. 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 assigned to a pre-selected color. This is of course a simple and straightforward solution, and is easy for observers to find a particular symbol or pattern. 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 dont use any predetermined color scheme, but calculate colors only based on the distance matrices, and rotate and flip colors to make them as smooth as possible. This strategy brings significant improvement to the alignment coloring, by showing the conserved regions and blocks in same or similar colors, while those irrelevant ones in quite different colors.

Since we are to convert the scaled coordinates to colors, choosing a suitable color space is an fundamental task. 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 found a problem, that some colors, like very dark ones and very light ones, did not perfectly serve the purpose of representing distances, yet made the graph more noisy. We realized that we dont really need the whole color space and all possible colors. Instead, those colors with proper range of lightness and different hues will be enough to do the job. So we chose to scale down to a two-dimensional space and map it to CIE Lab space with a fixed lightness value (75). 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). So two dimensions is a good balance.

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 (Nelder and Mead 1965), which is the default one, returns robust results but is relatively slow. CG (Fletcher and Reeves 1964) 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 (Belisle 1992) 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. 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. Algorithm Minimum penalty Aerage penalty Maximum penalty Standard deviation Initial 370,584 425,728 444,771 13,998 L-BFGS-B 136,909 178,850 247,218 17,871 L-BFGS-B - run 3 times 136,909 178,850 247,218 17,871 mlninb() 146,759 181,688 426,535 34,579 mlninb() - run 3 times 142,888 171,380 325,469 21,402

Obviously using L-BFGS-B is more stable than nlminb() on our test dataset.

References

[1] Robert C Edgar and Serafim Batzoglou. Multiple sequence alignment. *Current Opinion in Structural Biology*, 16(3):368–373, June 2006.

[2] JB Procter, J Thompson, I Letunic, C Creevey, F Jossinet, and GJ Barton. Visualization of multiple alignments, phylogenies and gene family evolution. *Nat Methods*, 7(3 Suppl):S16–25, 2010.

[3] Kuang Lin, Alex C. W. May, and William R. Taylor. Amino acid encoding schemes from protein structure alignments: Multi-dimensional vectors to describe residue types. *Journal of Theoretical Biology*, 216(3):361 – 365, 2002.

[4] J. D. Thompson, T. J. Gibson, and D. G. Higgins. Multiple sequence alignment using ClustalW and ClustalX. *Current protocols in bioinformatics/editoral board, Andreas D. Baxevanis...[et al.]*, pages Unit–2, 2002.

[5] Osnat Penn, Eyal Privman, Giddy Landan, Dan Graur, and Tal Pupko. An alignment confidence score capturing robustness to guide tree uncertainty. *Molecular Biology and Evolution*, 27(8):1759 –1767, 2010.

[6] 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*, 38(Web Server):W23–W28, May 2010.

[7] Ingwer Borg and Patrick J. F. Groenen. *Modern multidimensional scaling: theory and applications.* Springer, 1997.

[8] 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*, 92(9):338–341, September 1976.

[9] J. C. Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3-4):325–338, 1966.

[10] 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.

[11] R. Gentleman and Ihaka. R: A language for data analysis and graphics. *Journal Of Computational And Graphical Statistics*, 5(3):299–314.

[12] Francis Cailliez. The analytical solution of the additive constant problem. *Psychometrika*, 48(2):305–308, June 1983.

[13] Michael A. A. Cox and Trevor F. Cox. Multidimensional scaling. In *Handbook of Data Visualization*, pages 315–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[14] CIE. *Commission internationale de l'Eclairage proceedings, 1931*. [[Cambridge University Press]], Cambridge, 1932.

[15] &#32;Thomas Smith. The C.I.E. colorimetric standards and their use. *Transactions of the Optical Society*, 33(3):73–134, 1931.

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

[17] Ross Ihaka, Paul Murrell, Kurt Hornik, and Achim Zeileis. *colorspace: Color Space Manipulation*, 2009. R package version 1.0-1.

[18] Achim Zeileis, Kurt Hornik, and Paul Murrell. Escaping RGBland: Selecting colors for statistical graphics. *Computational Statistics & Data Analysis*, 53:3259–3270, 2009.

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

[20] D. Crockford. The application/json media type for javascript object notation (json). Technical report, RFC 4627, July, 2006.