APPROVAL SHEET

Title of Thesis: A web-based interactive visualization tool to explore patterns of amino acid substitution

Name of Candidate: Kamalika Das
Master of Science, 2005

Thesis and Abstract Approved: ______________________
Dr. Penny Rheingans
Associate Professor
Department of Computer Science and
Electrical Engineering

Date Approved: ____________________________

CURRICULUM VITAE

Name: Kamalika Das.

Permanent Address: 113-P, Dr.S.C.Banerjee Road, Calcutta 700010, India.

Degree and date to be conferred: Master of Science, August 2005.

Date of Birth: October 10, 1980.

Place of Birth: Calcutta, India.

Secondary education: South Point High School, Calcutta, 1997.

Higher Secondary education: South Point High School, Calcutta, 1999.

Collegiate institutions attended:
University of Maryland, Baltimore County, M. S. CS, 2005.
B.P.Poddar Institute of Management and Technology, India, B.Tech. CSE 2003.

Major: Computer Science.

Professional positions held:
Research Assistant, University of Maryland, School of Medicine (August 2005 -)
Software Design Engineer Intern at AGNIK, LLC (June 2005 - August 2005)
Research Assistant, CSEE Department, UMBC (January 2004 - January 2005).
Teaching Assistant, CSEE Department, UMBC (August 2003 - June 2004).

ABSTRACT

Title of Thesis: A web-based interactive visualization tool to explore patterns of amino acid substitution

Kamalika Das, Master of Science, 2005

Thesis directed by: Dr. Penny Rheingans

Among all the natural sciences, biology is the most data-rich subject. A few central databases of biology are growing exponentially in size with time. There has also been an exponential increase in the diversity of specialized databases in the last decade. These vast repositories of data contain valuable information relevant to cutting-edge biological research, such as research on genetic evolution. In their attempts to explain evolution, biologists, to date, have relied on observations alone to explain the divergence of protein sequences to form different types of organisms. Since amino acids are the basic building blocks of protein molecules, understanding the effect of the properties of amino acids in the substitution process might give biologists insight into genetic evolution. In this research, we represent the relationship among the different amino acids and their physiochemical properties using a graph-based visualization technique. We have developed a number of new visualization schemes for aiding the understanding of the relationship between amino acids' properties and amino acid substitution patterns so that biologists can explain the observations in genetic evolution in the form of a scientific cause-and-effect model.

A web-based interactive visualization tool to explore patterns of amino acid substitution

by
Kamalika Das

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
2005

Dedicated to Ma, Appa, Amy, and Kanishka

Contents

1 Introduction 1

2 Biological Background 5

2.1 Genes and Evolution 6

2.2 Amino Acid Data 8

2.3 Application 13

2.4 Design 14

2.4.1 2D vs. 3D Visualization 15

2.5 User Interface 18

3 Related Work 20

3.1 Graph Visualization 21

3.2 Biovisualization 25

4 Minimum Spanning Tree Representation 27

4.1 Spanning Tree Generation 28

4.2 Spanning Tree Layout 29

4.3 Index Spanning Tree 34

4.3.1 Banded Edges 37

4.4 Matrix Spanning Tree 42

4.5 Evaluating the Spanning Tree Visualization 45

5 Comparing Spanning Trees **54**

5.1 Layout Algorithms 56

5.1.1 Joint Independent Layout 56

5.1.2 Joint Layout Influenced by the Index Tree 58

5.1.3 Joint Layout Influenced by the Matrix Tree 64

5.2 Evaluation of the Three Joint Layouts 66

6 Sparkler Arm Visualization Technique **71**

6.1 Multiple Query Visualization 71

6.2 Sparkler Arm Technique 72

6.3 Usefulness of the Sparkler Arm View 74

7 Research Contribution and Future Work **76**

7.1 Research Contribution in Biology 76

7.2 Research Contribution in Visualization 77

7.3 Future Work 78

7.4 Conclusion 80

Bibliography **81**

List of Figures

1.1 Exponential growth of central databases in biology. (Reproduced, with permission, from Dr. Stephen Freeland's slides, UMBC.) 2

2.1 Description of components of genetic code. (Image courtesy of Dr. Stephen Freeland [16].) 7

2.2 Standard genetic code. (Ref: http://evolvingcode.net/web-introduction.php) 8

2.3 BLOSUM45 Substitution Matrix, Authors:Henikoff, S. and Henikoff, J.G [Ref: http://www.evolvingcode.net:8080/aaindex/] 12

2.4 Existing work influencing this research: (a) spanning tree generated by Tomii and Kanehisa [43], (b) the webtool developed by Blazej Bulka to reconstruct the results of Tomii et al. 14

2.5 The three basic views of the system: (a) the index spanning tree view, (b) the joint layout view, (c) the Sparkler Arm view 16

2.6 Graphical User Interface of the System 19

4.1 Random layout of the MST with 40 indices 30

4.2 Modified Reingold-Tilford layout of the MST with 40 indices 31

4.3 Variation of spring constant (a) $ks = 500$ (b) $ks = 5$ 32

4.4 Variation of repulsive force constant (a) $kr = 50$ (b) $kr = 150000$ 34

4.5 Graphical user interface for displaying the database of amino acid indices. It gives the user the option to select individual indices for display 35

4.6 Minimum spanning tree from 40 selected indices 36

4.7 Details of Edge Lengths Shown in Text Area for the Index Spanning Tree . 37

4.8 Banded edges in index spanning tree and color index as part of the graphical user interface 39

4.9 Banded edges in the index spanning tree with differences of values of amino acids between the two associated properties shown as bars perpendicular to the blocks. The length of the perpendicular bars is proportional to the difference value. 40

4.10 Banded Edges in Index Spanning Tree with α-value mapping of color . . . 41

4.11 Graphical user interface for displaying the database of similarity matrices. It gives the user the option to select individual matrices for display 42

4.12 Minimum Spanning Tree of 45 Similarity Matrices 43

4.13 Details of Edge Lengths Shown in Text Area for the Matrix Spanning Tree . 44

4.14 Experimental Results Showing the Total Error for Different Sizes of the Index Spanning Tree 49

4.15 Experimental Results Showing the Total Error for Different Sizes of the Matrix Spanning Tree 50

4.16 Experimental results showing the average difference in lengths of the edges representing the minimum weight and the second lowest weight. The red graph represents the average of the absolute difference whereas the green graph represents that of the exact difference. 52

4.17 Experimental results showing the average difference in lengths of the edges representing the minimum weight and the second lowest weight. The red graph represents the average of the absolute difference whereas the green graph represents that of the exact difference. 53

5.1 Joint independent layout where positions of matrix nodes are determined by spring embedding 59

5.2 Joint layout influenced by index tree for threshold=0.5 61

5.3 Joint layout influenced by index tree for threshold=0.58 62

5.4 Joint layout influenced by index tree for threshold=0.75 63

5.5 Joint layout influenced by index tree for threshold=1.0 64

5.6 Joint layout influenced by index tree for threshold=0.5 65

5.7 Experimental result showing the error between the first five indices (taken in sequence) with increasing number of matrices for the joint independent layout 67

5.8 Experimental result showing the error between the first five indices (taken in sequence) with increasing number of matrices for the joint layout influenced by Index tree 69

5.9 Experimental result showing the error between the first five indices (taken in sequence with increasing number of matrices for the joint layout influenced by Matrix tree 70

6.1 Sparkler Arm View with A (Alanine) as the chosen Amino Acid and 10 Mutation Matrices 73

List of Tables

2.1 Index: Molecular Weight, Authors: Fasman, G.D., Index Number: 72 (Ref: [10]) 10

Chapter 1

Introduction

“The purpose of computing is insight, not numbers.” - R. Hamming, 1973.

Information visualization allows the user to visualize large data sets efficiently. Often the information gathered from large repositories or databases is abstract and difficult to comprehend and use. Information visualization aids the process of interpreting these large data sets by abstracting the raw data and representing it in a user-friendly graphical way. Like good writing, good graphical displays make communication of ideas lucid, precise, and efficient. On the other hand, bad graphical displays may distort or obscure information, making it difficult to understand or analyze, thereby reducing the communicative effectiveness. Efficient graphical displays give an overview of all relationships existing between different entities in a data set, help focus attention on the important aspects of the relational model, and show the details only on demand [6].

Computational biology is currently a fast-growing interdisciplinary area with ground-breaking

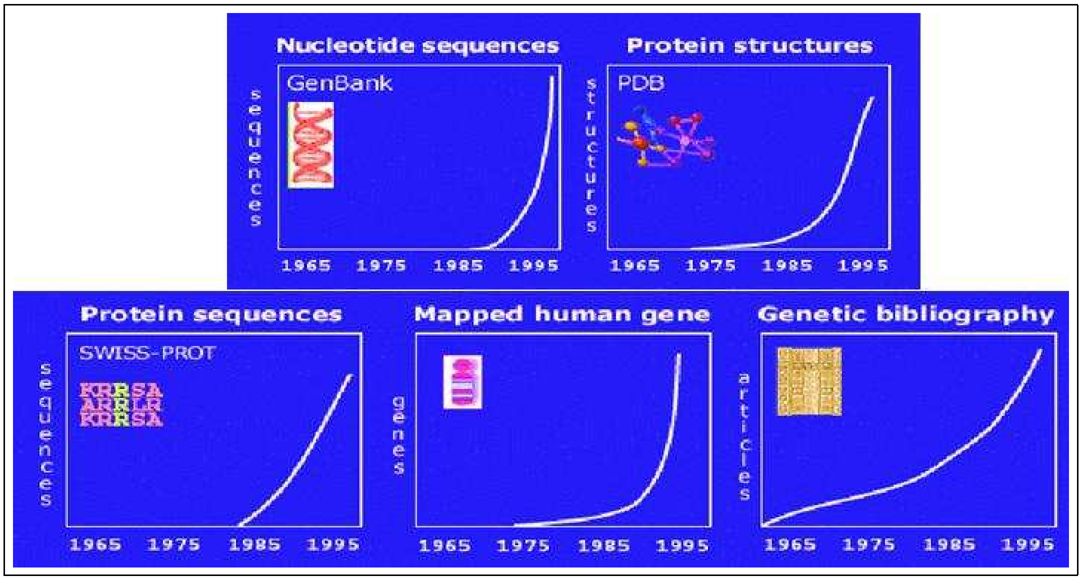


Figure 1.1: Exponential growth of central databases in biology. (Reproduced, with permission, from Dr. Stephen Freeland's slides, UMBC.)

contributions from both computing sciences and biology. Biologists all over the world are trying to explain the process of evolution based on available data from several subfields of biology. Biology is an extremely data-rich subject. A number of central databases in biology, such as the protein sequence database [9] and the human genetic mapping database [38], are growing exponentially with time. Figure 1.1 shows the growth of such databases over a period of 30 years from 1965 to 1995. This gives us an estimate of the amount of data that bioinformatics deals with. There has also been an exponential increase in the diversity of specialized databases. InterPro (an integrated resource of protein families, domains and functional sites) and NDB (nucleic acid-containing structures) are examples of such specialized databases. The current count of biological databases is 548, which is 162 more than the number that existed a year before [17]. Representing these vast repositories of information in a comprehensible manner is a challenging task and so one major area of research in information visualization is bioinformatic visualization.

The objectives of visualizing the elements of these databases can vary. Sometimes the observer wants to understand the overall relationship between biologically similar entities belonging to one or multiple databases. Alternatively, the visualization can be an attempt to find out biologically significant information such as chemical properties of different entities and how these properties are responsible for shaping the existing theories of biology. The choice of the visualization technique depends on the goal of the observer. Different techniques have different advantages with respect to the amount or type of information that is being conveyed to the observer.

The majority of current bioinformatic visualization systems concentrate on easily understandable representation of experimental data, such as comparison of evolutionary trees [32], visualization of exact geometric structures of double-helix DNA [22], and 3D visualization of protein molecules. These systems help in visualizing DNA sequences, annotating the results generated by BLAST (Basic Logical Alignment Search Tool) [5] searches, and facilitating protein alignment. However, the emphasis of most existing systems has been on the visualization of geometrical structures of proteins. To date, little importance has been attached to the actual study of the physical and chemical properties of amino acids, which would enable biologists to explain genetic evolution.

To overcome this lacuna, we developed a visualization tool that displays amino acids and their properties as a relational or causal model. Graphs are ubiquitous models for representing large groups of similar or related objects. They efficiently model many real-world applications. The choice of the graph layout entails the optimization of its geometric and topological objectives [13]. We use different graphs for visualizing different hierarchical relationships between properties of amino acids and how these properties affect the arrangement of amino acids to form long chains of polypeptides, which are more commonly known as proteins. For visualizing the relationships between amino acid pairs, we use a minimum spanning tree representation. A similar view is also used for showing the similarity between different amino acids. For understanding how the properties of amino acids have influenced the similarity between them, we develop a number of new layout algorithms where two minimum spanning trees are compared based on their joint layout. Finally, we define a *sparkler arm* view for aligning different amino acids with respect to their distances from a specific chosen one.

Chapter 2 provides the necessary biological motivation for understanding this work. Chapter 3 discusses relevant related work. Chapters 4, 5, and 6 present the detailed approach for each of the three visualization techniques that have been used in the system. Chapter 7 highlights the research contribution of this work in the fields of information visualization and biology, as well as the results of this research and possible future work in this area.

Chapter 2

Biological Background

“Many biological problems are too complex to be solved by biologists alone,” says biologist Gosta Nachman of Copenhagen University [33]. Biological experiments result in a huge quantity of numerical data. This data contains valuable information relevant to cutting-edge biological research. However, it is difficult for biologists to manually process such a large quantity of numerical data to extract necessary information. Computer scientists make this task easier by applying different data analysis techniques and transforming the data to easily representable and comprehensible formats. Bioinformatics is the application of computational sciences to facilitate research in biological sciences [4]. Therefore, a basic understanding of the fundamental processes and theories of biology is essential for appreciating this research work in bioinformatic visualization. Currently, biologists are trying to explain the process of evolution by exploring the physical and chemical properties of amino acids which are the building blocks of protein molecules in any living organism.

2.1 Genes and Evolution

Genetic information is stored in the DNA of an organism. It is the encoded information in the DNA that distinguishes one living being from another. It can be described as a set of ‘how to’ instructions for building an organism. A DNA molecule encodes one or more genes; a gene encodes a single protein, which is composed of a number of amino acids. The language-based terminology reflects the fact that both genes and proteins are essentially 1-dimensional arrays of chemical letters. Each DNA is composed of four nucleotides, which are abbreviated as **A**, **C**, **T**, and **G**. The DNA nucleotides are transformed, by a process called transcription, to mRNA nucleotides abbreviated as **A**, **C**, **U**, and **G**. Proteins, on the other hand, are built from twenty different amino acids [16][47]. Figure 2.1 describes the structures of the mRNA nucleotides. The figure also shows the chemical structures of these 20 amino acids built on these nucleotide bases.

Given the difference in size between the DNA and the protein alphabets, it is clear that there cannot be a simple 1:1 correspondence between the nucleotides of a mRNA strand and the amino acids of a protein. Triplets of mRNA nucleotides called *codons* translate into a single amino acid. For example, the triplet **GGG** corresponds to the amino acid Glycine (*Gly*) while **ACG** corresponds to the amino acid Serine (*Ser*). There is a standard genetic code for this translation. The standard genetic code is a set of rules for a many-to-one translation of the codons to the amino acids. Figure 2.2 shows the codon-amino acid pairs that exist. The uppercase triplets such as **UUU** or **GUC** denote the codons, whereas the other three-lettered groups such as *Phe* and *Val* correspond to amino acid abbreviations.

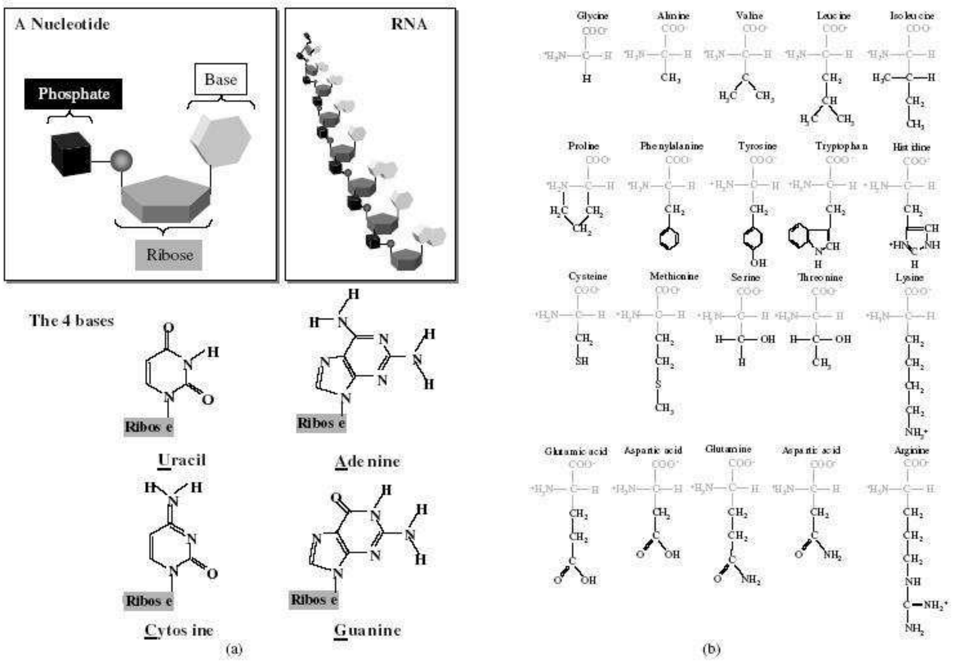
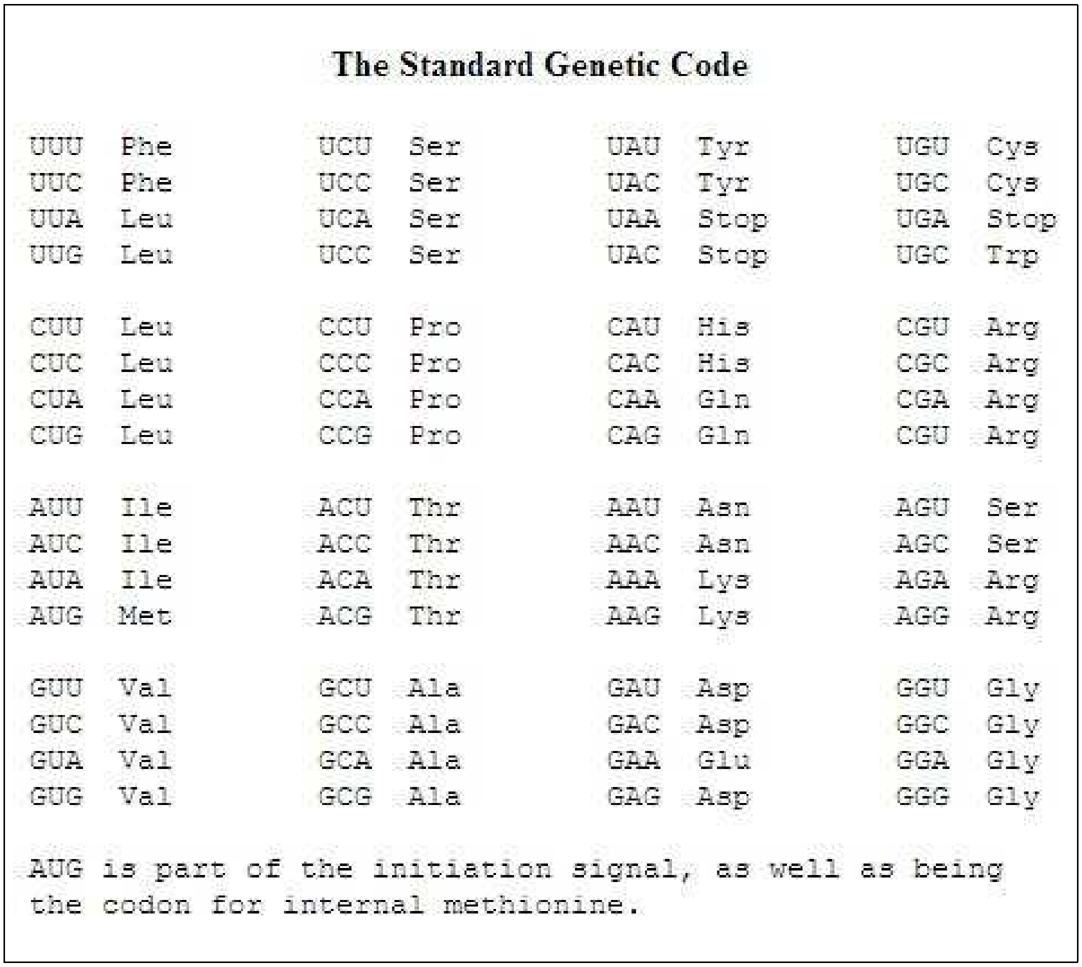


Figure 1. The components of the genetic code: (a) genetic information is stored as a linear array of polymerized nucleotides (i.e. nucleic acid). The genetic alphabet comprises 4 different nucleotides: 2 single ring pyrimidines (C and U for RNA), and 2 double ring purines (A and G); (b) The protein alphabet of 20 amino acids. Genes store the information to make proteins, which perform the biochemical catalysis of life. Proteins are polymerized linear arrays of amino acids.

Figure 2.1: Description of components of genetic code. (Image courtesy of Dr. Stephen Freeland [16].)

Biologists, in their attempt to explain evolution are trying to answer a few fundamental questions including

- How has a standard genetic code evolved over time?
- How has the process of evolution caused protein sequences to diverge?
- How can the understanding of evolution as a cause of protein sequence divergence help in the prediction of protein structures?
- How can evolution explain the varied functions of genes and proteins?
- How can the understanding of the standard genetic code help in automating the process of recognizing genes in a new genome?



The Standard Genetic Code

UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys
UUC	Phe	UCC	Ser	UAC	Tyr	UGC	Cys
UUA	Leu	UCA	Ser	UAA	Stop	UGA	Stop
UUG	Leu	UCC	Ser	UAC	Stop	UGC	Trp
CUU	Leu	CCU	Pro	CAU	His	CGU	Arg
CUC	Leu	CCC	Pro	CAC	His	CGC	Arg
CUA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CUG	Leu	CCG	Pro	CAG	Gln	CGU	Arg
AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser
AUC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
AUA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
AUG	Met	ACG	Thr	AAG	Lys	AGG	Arg
GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly
GUC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GUA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GUG	Val	GCG	Ala	GAG	Asp	GGG	Gly

AUG is part of the initiation signal, as well as being the codon for internal methionine.

Figure 2.2: Standard genetic code. (Ref: http://evolvingcode.net/web-introduction.php)

- How can this study help researchers to build models of evolutionary relatedness?

For answering these questions, biologists have a plethora of information available on amino acids and their physiochemical properties.

2.2 Amino Acid Data

Amino acids are molecules consisting of a single central carbon atom with four groups attached to its four valencies. These groups are a hydrogen (H) atom, an amino (NH_2)

group, a carboxyl (COOH) group, and a ‘R’ group. The structure of the R group determines which amino acid it is. There are 20 known amino acids in the genetic code. Shuichi Kawashima and Minoru Kanehisa [26] first represented the different properties of amino acids in the AAIndex database. The AAIndex database can be accessed at http://www.genome.ad.jp/aaindex. An amino acid index is a property (physiochemical or biochemical) of the amino acid. Each index is a 20-element vector; each element of the vector is the value of that property for one of the 20 amino acids. The AAIndex database consists of two parts: the first part is the index database and the second part is the mutation matrix database. A similarity matrix, or mutation matrix, is a set of 20×20 numerical values calculated on the basis of the observed frequency with which one amino acid replaces another over the course of evolution. These matrices are used for protein sequence alignments and similarity searches [5]. The index and matrix data that we have used for our system is taken from the “UMBC AAIndex Database” [10]. Blazej Bulka, a member of the evolvingcode team at UMBC Biological Sciences department, has constructed this database by modifying the original AAIndex database. The UMBC AAIndex database also consists of two parts, namely, the indices and matrices. The version of the UMBC AAIndex database that we are using for this research has 494 indices and 83 mutation matrices.

The indices represent physiochemical properties of amino acids. The range of values for each property varies across the set of indices. To facilitate computations, all the indices in the UMBC AAIndex database are linearly normalized with 0 corresponding to the minimum observed value and 1 corresponding to the highest. Therefore, the amino acid index data that we have used in our computations could have a value between 0 and 1. However,

Amino acid	3-letter code	Symbol	Normalized value
Alanine	*Ala*	**A**	0.108
Arginine	*Arg*	**R**	0.767
Asparagine	*Asn*	**N**	0.442
Aspartic acid	*Asp*	**D**	0.449
Cysteine	*Cys*	**C**	0.357
Glutamine	*Gln*	**Q**	0.550
Glutamic acid	*Glu*	**E**	0.557
Glycine	*Gly*	**G**	0.0
Histidine	*His*	**H**	0.620
Isoleucine	*Ile*	**I**	0.434
Leucine	*Leu*	**L**	0.434
Lysine	*Lys*	**K**	0.550
Methionine	*Met*	**M**	0.574
Phenylalanine	*Phe*	**F**	0.698
Proline	*Pro*	**P**	0.310
Serine	*Ser*	**S**	0.232
Threonine	*Thr*	**T**	0.341
Tryptophan	*Trp*	**W**	1.0
Tyrosine	*Tyr*	**Y**	0.821
Valine	*Val*	**V**	0.326

Table 2.1: Index: Molecular Weight, Authors: Fasman, G.D., Index Number: 72 (Ref: [10])

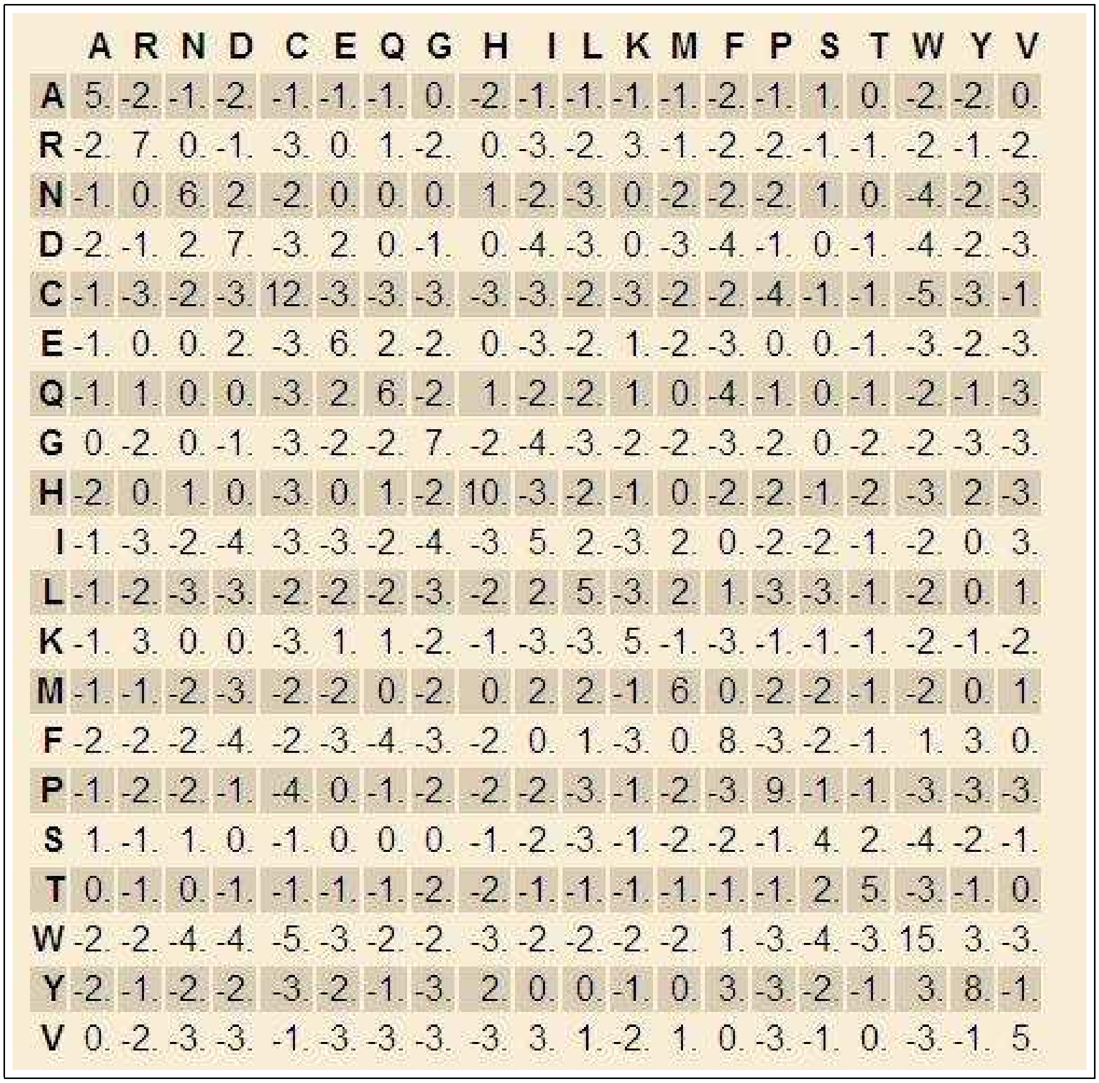
if the distribution of values of a property across the 20 amino acids is not linear, then the linear normalization would fail to capture the nature of the distribution and might not yield very accurate results.

One property (molecular weight) or amino acid index is shown in Table 2.1. The first column of the table contains the name of the amino acid. The second column shows a three-letter representation of the amino acid; the third column shows the symbol of the amino acid; and the last column shows the value of the property for that amino acid. We notice here that amino acid Glycine (*Gly*) has a very low molecular weight compared to Arginine (*Arg*).

The similarity or mutation matrices are 20×20 matrices where both the rows and columns correspond to the amino acids. These matrices represent the observed “similarity” between pairs of amino acids under different conditions and in different units of measurements. By similarity we mean the tendency of one amino acid replacing another during evolution. The matrix corresponds to a Markovian model of evolution in which amino acids mutate randomly and independently from one another but according to some predefined probabilities. A positive value of the (i,j)th element of the mutation matrix means that the observed frequency of substitution of the amino acid at row i by the one at column j is more than the probabilistic prediction, a negative value indicates that the observed frequency is less than the expected frequency and the value of 0 indicates that the observed frequency is the same as the predicted frequency. A diagonal element represents the chance that an amino acid is not substituted.

One such similarity matrix is shown in Figure 2.3, displaying the BLOSUM45 substitution matrix used in the bioinformatics tool BLAST [5]. It can be seen that amino acid **R** (Arginine or *Arg*) replaces **I** (Isoleucine or *Ile*), and vice versa, much less often than expected, whereas the frequency of substitution between **N** (Asparagine or *Asn*) and **D** (Aspartic Acid or *Asp*) is quite high. The diagonal elements of the matrix have very high values, illustrating that the most common observation is that the amino acids do not get substituted very frequently.

This research aims to help biologists to answer questions on evolution by using this data on amino acids. We introduce a meaningful way of displaying the relationships that exist among different indices and among different mutation matrices. The design allows the



	A	R	N	D	C	E	Q	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	5.	-2.	-1.	-2.	-1.	-1.	-1.	0.	-2.	-1.	-1.	-1.	-1.	-2.	-1.	1.	0.	-2.	-2.	0.
R	-2.	7.	0.	-1.	-3.	0.	1.	-2.	0.	-3.	-2.	3.	-1.	-2.	-2.	-1.	-1.	-2.	-1.	-2.
N	-1.	0.	6.	2.	-2.	0.	0.	0.	1.	-2.	-3.	0.	-2.	-2.	-2.	1.	0.	-4.	-2.	-3.
D	-2.	-1.	2.	7.	-3.	2.	0.	-1.	0.	-4.	-3.	0.	-3.	-4.	-1.	0.	-1.	-4.	-2.	-3.
C	-1.	-3.	-2.	-3.	12.	-3.	-3.	-3.	-3.	-3.	-2.	-3.	-2.	-2.	-4.	-1.	-1.	-5.	-3.	-1.
E	-1.	0.	0.	2.	-3.	6.	2.	-2.	0.	-3.	-2.	1.	-2.	-3.	0.	0.	-1.	-3.	-2.	-3.
Q	-1.	1.	0.	0.	-3.	2.	6.	-2.	1.	-2.	-2.	1.	0.	-4.	-1.	0.	-1.	-2.	-1.	-3.
G	0.	-2.	0.	-1.	-3.	-2.	-2.	7.	-2.	-4.	-3.	-2.	-2.	-3.	-2.	0.	-2.	-2.	-3.	-3.
H	-2.	0.	1.	0.	-3.	0.	1.	-2.	10.	-3.	-2.	-1.	0.	-2.	-2.	-1.	-2.	-3.	2.	-3.
I	-1.	-3.	-2.	-4.	-3.	-3.	-2.	-4.	-3.	5.	2.	-3.	2.	0.	-2.	-2.	-1.	-2.	0.	3.
L	-1.	-2.	-3.	-3.	-2.	-2.	-2.	-3.	-2.	2.	5.	-3.	2.	1.	-3.	-3.	-1.	-2.	0.	1.
K	-1.	3.	0.	0.	-3.	1.	1.	-2.	-1.	-3.	-3.	5.	-1.	-3.	-1.	-1.	-1.	-2.	-1.	-2.
M	-1.	-1.	-2.	-3.	-2.	-2.	0.	-2.	0.	2.	2.	-1.	6.	0.	-2.	-2.	-1.	-2.	0.	1.
F	-2.	-2.	-2.	-4.	-2.	-3.	-4.	-3.	-2.	0.	1.	-3.	0.	8.	-3.	-2.	-1.	1.	3.	0.
P	-1.	-2.	-2.	-1.	-4.	0.	-1.	-2.	-2.	-2.	-3.	-1.	-2.	-3.	9.	-1.	-1.	-3.	-3.	-3.
S	1.	-1.	1.	0.	-1.	0.	0.	0.	-1.	-2.	-3.	-1.	-2.	-2.	-1.	4.	2.	-4.	-2.	-1.
T	0.	-1.	0.	-1.	-1.	-1.	-1.	-2.	-2.	-1.	-1.	-1.	-1.	-1.	-1.	2.	5.	-3.	-1.	0.
W	-2.	-2.	-4.	-4.	-5.	-3.	-2.	-2.	-3.	-2.	-2.	-2.	-2.	1.	-3.	-4.	-3.	15.	3.	-3.
Y	-2.	-1.	-2.	-2.	-3.	-2.	-1.	-3.	2.	0.	0.	-1.	0.	3.	-3.	-2.	-1.	3.	8.	-1.
V	0.	-2.	-3.	-3.	-1.	-3.	-3.	-3.	-3.	3.	1.	-2.	1.	0.	-3.	-1.	0.	-3.	-1.	5.

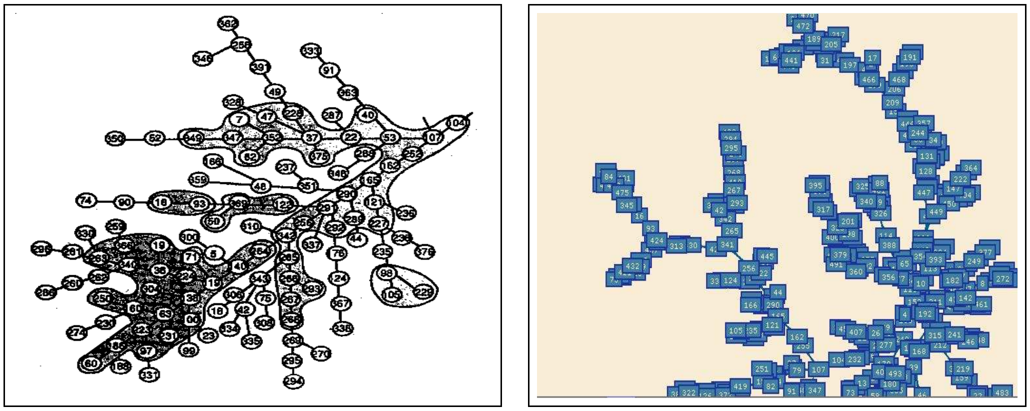
Figure 2.3: BLOSUM45 Substitution Matrix, Authors:Henikoff, S. and Henikoff, J.G [Ref: http://www.evolvingcode.net:8080/aaindex/]

user to analyze the influence of a group of indices on a group of mutation matrices. The visualization tool also allows the user to explore the amino acids based on their distances (similarities) from a specific amino acid, thus linking observed evolution to the physics behind it.

2.3 Application

Researchers have developed a number of visualization tools in bioinformatics to represent different types of biological data such as the double helix DNA strands, 3D protein molecule structures and 3D amino acid sidechains. However, the task that this visualization tries to accomplish is different from the usual structure-visualization task. In this research, following the work of Tomii and Kanehisa [43], we try to connect the existing amino acid properties represented by the amino acid indices to their observed substitution patterns represented by the mutation matrices. The dataset that we are looking at is not very large. However, we are attempting to fit this data into a cause-and-effect model where we attribute the relatedness of pairs of amino acids to their physical and chemical properties. This study is interesting because other than the obvious utility in helping to understand the origins and evolution of life, it will give the biologists an insight into the process of protein formation which might help in designing new protein molecules for better drug design.

Before we go into the details of our system, we discuss the work of Tomii and Kanehisa [43], which directly inspires this research. Tomii and Kanehisa prepared a single-linkage hierarchical cluster analysis of the amino acid indices in the AAIndex database. They also did a similar cluster analysis for the mutation matrices in the database and studied the relationships between them. They reconstructed these matrices from the combination of amino acid indices in order to find which properties of amino acids are reflected most. In this work, the basic idea is that the diversity of amino acid properties is the key to the structure, function and evolution of protein molecules, which is the main theme behind



(a) **(b)**

Figure 2.4: Existing work influencing this research: (a) spanning tree generated by Tomii and Kanehisa [43], (b) the webtool developed by Blazej Bulka to reconstruct the results of Tomii et al.

our research as well. Existing work in this area include a web-based visualization tool developed by Blazej Bulka at University of Maryland, Baltimore County. This tool allows the users to visualize relations among amino acid properties, and relate them to the mutation matrices. It also allows recreation of the results of cluster analysis described by Tomii and Kanehisa. Figure 2.4 shows the results generated by Tomii and Kanehisa.

2.4 Design

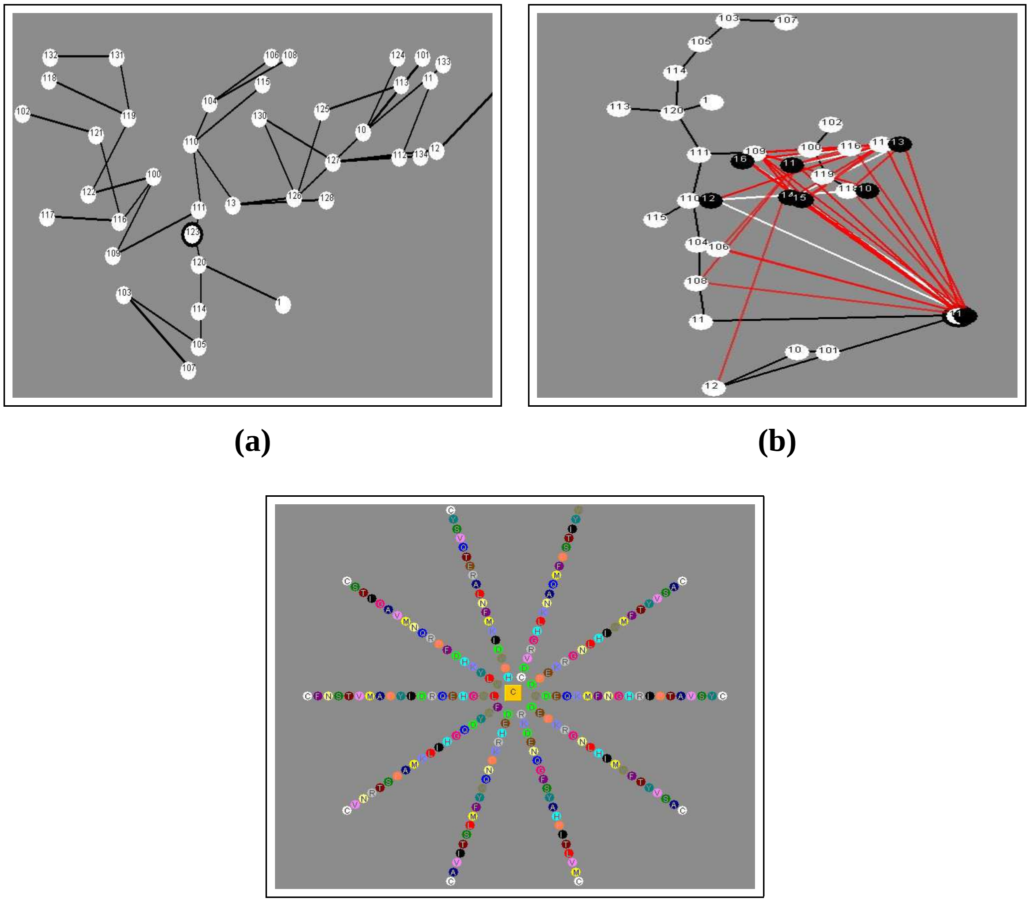
Higher-dimensional visualization techniques are confusing for the human perception system and can sometimes be misleading. Therefore, the representation should aim at reducing the dimensionality of the data and representing the important characteristics in terms of a few salient features. An effective and intuitive representation would be one with the lowest possible dimension with no ambiguity in representation of either the data or their relationships. Interactivity is needed to control the level of detail of the information being displayed. Since these biological databases are growing rapidly in size, scalability of the system is an important issue that needs to be considered at the design phase.

While designing the representation scheme, we keep in mind the desired characteristics of the system, which can be summarized as:

- effective representation of high-dimensional data
- intuitive and interactive visualization technique well suited for the purpose
- unambiguous data representation
- appropriate choice of mathematical or statistical methods to suit the graphical model
- scalability of the system

2.4.1 2D vs. 3D Visualization

Considering the design requirements discussed above, we chose to reduce the dimensionality of the representation from twenty to either two or three. Tree visualization can be either 2D or 3D. If a graph becomes too large, then a conventional 2D graph layout algorithm might fail to scale up. In that case, 3D visualization can help in reducing the occlusion of the layout due to the use of the third axis. Most 2D layout algorithms have been extended to a 3D case in a hope to find more space for representing large data sets in the extra dimension that is added. However, almost all the challenges in graph visualization in two dimensions are also present in 3D graph visualization, such as lack of efficient navigation techniques, edge crossing minimization for large data sets, and finding an occlusion-free



(a) **(b)**

(c)

Figure 2.5: The three basic views of the system: (a) the index spanning tree view, (b) the joint layout view, (c) the Sparkler Arm view

view of a particular portion of the graph. For this thesis, we concentrate on 2D visualization of graphs, because the data set that we are working with is not very large and can easily be represented in the 2D screen space without occlusion problems. The 3D graph algorithms also have higher time complexity and would make the system slower.

We designed three fundamental views that serve multiple purposes in terms of understanding the relationships and analyzing the data. Figure 2.5 shows three basic views of the system. The views are:

- a single spanning tree view,
- a comparative view of two spanning trees, and
- a sparkler-arm view

The *spanning tree view* represents the relationships between similar elements of the database. There are two types of spanning trees: the index spanning tree and the matrix spanning tree. The index spanning tree depicts the relationships among the 494 different properties of amino acids in the amino acid index database. The matrix spanning tree represents the relationships among the 83 similarity matrices or mutation matrices.

The second *comparative view* allows the user to compare the two spanning trees (i.e. the index and matrix spanning trees). For this representation, we compute the individual minimum spanning trees and then compute the distances of each index node from each of the matrix nodes. The distance metric is based on the numerical values of the corresponding index and the matrix as specified in the AAIndex database. We refer to this distance as the computed distance of an edge connecting two nodes of the two trees. We choose a threshold, displaying only those edges which represent a weight less than that threshold. This draws the attention of the user to the most significant connections between the two trees. This view allows the user to understand how the amino acid indices influence the observed values of the similarity matrices.

In the *sparkler-arm view*, we explore the relationships among the twenty amino acids. Here, the user is given the option of selecting an amino acid and a few similarity matrices. Then we represent the remaining nineteen amino acids in increasing order of their computed distances from the selected amino acid for each one of the selected matrices. Each arm of the sparkler corresponds to a similarity matrix.

2.5 User Interface

Our visualization tool has an efficient and user-friendly graphical user interface supporting many functions. It supports a number of navigation functions such as zoom in, zoom out, and graph panning by dragging individual nodes. It is interactive, allowing the user to choose the details that are to be displayed in any particular view. It allows the user to either view the entire database or select a group of nodes for display. The user can select nodes either by checking individual rows of indices or matrices in a table or by specifying a keyword for selection. It also allows the user to add new indices to the system for visualization. Figure 2.6 shows the user interface design. The first button lets the user read the database of indices or matrices and select the appropriate ones for display. It has slider bars for on-the-fly modification of system parameters such as visualization thresholds and a color index for easy reference of the color-coded amino acids. The graph information section is the text area that displays information about selected entities when clicked. This view helps us find similar mutation matrices.

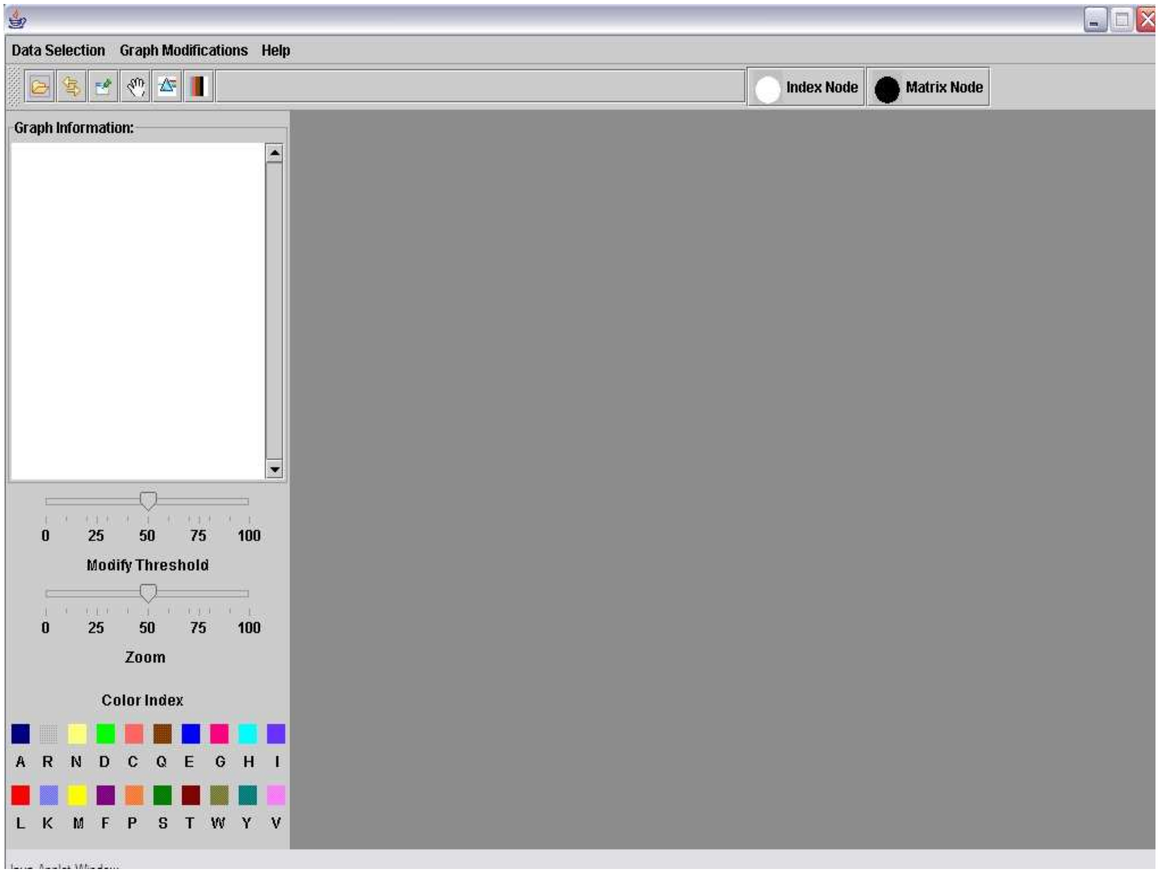


Figure 2.6: Graphical User Interface of the System

Chapter 3

Related Work

The fields of computational biology and bioinformatics have experienced rapid, groundbreaking advancement in the last decade. A number of visualization tools have been developed to aid in the explanation of biological theories. These tools use different visualization techniques, since each is tailored to serve the purpose for which it was built. The visualization schemes depend on the amount of data to be represented, the dimensionality of the data, and the nature of the information to be extracted out of the data.

Before reviewing the existing visualization tools and techniques, we first discuss the tool that we are building and the techniques that we are using for the purpose. Our visualization tool is a combination of multiple visualization techniques such as graph-based visualization and the sparkler-arm visualization. In the spanning tree view of the system, we use different graph visualization techniques to represent the spanning tree in the most meaningful way. In the joint layout of the index and the matrix spanning trees, we have experimented with a number of new layouts, which are modifications of standard graph layout algorithms.

In the following sections we review the existing graph visualization techniques and discuss our choices of specific techniques and modifications to those techniques to fit our requirements. We also review the broad categories of bioinformatic visualization and analyze the novelty of this tool in the existing scenario.

3.1 Graph Visualization

The amino acid data can be classified as a high-dimensional relational data model, which can be represented using a graph. Depending on the type of data to be represented, an appropriate graph structure is chosen: general graphs, which can be drawn by either a force-directed approach or a multi-dimensional approach; planar graphs; directed acyclic graphs (DAGs); or trees.

There exist a number of efficient graph visualization techniques in two-dimensional or three-dimensional space [24]. Depending on the nature of the system, different optimization criteria are chosen for the graph structure being used. These optimization factors include edge bends or edge crossings, and algorithmic efficiency. Other important factors to take into consideration while constructing a graph are the size of the graph and consistency of the graph-structure in conveying information.

The force-directed method of graph drawing is based on the idea of a spring model. In this technique, the nodes are placed randomly in the screen space. A repulsive force between every pair of nodes and an attractive force along each edge is computed, and the resultant net force is calculated. Based on the net force, the final positions of the nodes are determined using gradient descent. Tutte's barycenter algorithm [12], spring-embedding algorithms [13], and gravitational force algorithms [13] are examples of existing force-directed algorithms. Spring-embedded graphs are commonly used in representing graphs such as telecommunication networks, where the number of nodes and edges is large. The force-directed method is particularly useful when the edges of the graph can be assigned weights based on the properties they represent.

Orthogonal graph drawing is a technique for generating planar graphs [14]. Applications such as database diagrams, software engineering data-flow diagrams, and electrical circuit-diagrams follow this pattern of graph drawing. Orthogonal graph drawing uses polylines to represent edges. This means that each edge in a graph is broken into a chain of vertical and horizontal straight lines. Optimization entails identifying a layout that uses uniform (short) straight lines and that minimizes bends and crossings.

Hierarchical or upward line drawing results in a directed acyclic graph, which is used to represent a hierarchical relationship between the different objects represented as nodes in the graph. In this representation, the nodes are constrained to lie on different horizontal layers, depending on the hierarchical relationship they have with their immediate neighbors. Sugiyama et al. [42] describe a hierarchical graph drawing algorithm for drawing DAGs.

Other ways of representing high-dimensional data are relational perspective mapping [29], hyperbolic-plane mapping [46], and hierarchical topographic visualizations [25]. All these methods have been developed from a theoretical graph model perspective. They consist of a number of intermediate steps and have very high time complexity that is exponential in the number of nodes. Therefore, unless the relative distances of the high-dimensional data set are important, these algorithms are not used for graph layout.

A tree is a specialized graph without any cycles. A classical tree layout algorithm by Reingold and Tilford positions the child nodes below the parent node [37]. This algorithm lays out the graph in horizontal layers recursively, considering each subtree as a tree with the topmost node in the subtree being considered as the root of that tree. The root is always located above both of its children. This algorithm runs in linear time (in number of nodes). However, it requires prior knowledge of the parent-child relationship that exists between the nodes. However, in a spanning tree representing relationships between entities, there does not exist a hierarchical relation among nodes, therefore no node can be designated deterministically as the root. Other tree layouts that exist are H-tree layouts for binary trees [41], radial trees [15] and cone trees [31]. H-trees are grid embeddings of free trees [15], which do not have a left-to-right ordering of nodes. In this technique, the nodes are placed on the grid points and the edges are routed along the horizontal and vertical grids. Therefore, this algorithm can only lay out binary trees. Radial tree drawing assumes the same hierarchical structure as the Reingold-Tilford algorithm [37], but draws nodes at the same level of hierarchy at the same radial distance from the root node. Cone trees are generally used for 3-dimensional data and the hierarchical structure is still used.

Several tree visualization techniques are tailored to serve different requirements of biologists studying genetic evolution. TreeJuxtaposer [32] is a tool for comparing very large phylogenetic (evolutionary) trees. TreeJuxtaposer associates each node in one tree with a corresponding similar node of the other tree. The nodes in the two trees that do not have corresponding matches are the ones that relate to the structural difference between the two trees. By studying such structural differences in evolutionary trees, biologists are able to answer questions about the evolutionary process. Another phylogenetic tree comparison tool is the Tree Set Visualization Program [2]. This program represents a tree space generated by comparing two trees using certain distance metrics using multi-dimensional analysis.

The above tools are undoubtedly suited for the application they are built for. They can effectively display a very large number of nodes without the user losing focus or context. However, our application does not need a display for supporting thousands of nodes. On the other hand, in our application we compare two spanning trees, the nodes of which represent different entities, and in which the number of nodes in the two trees might be different. Therefore, these tree comparison tools would not serve our purpose of visualization, because they are based on the assumption that the trees being compared represent similar objects and have approximately the same number of nodes. All the graph drawing algorithms described here are used for different visualization applications. Some of these algorithms are suitable for small number of nodes (of the order of a few hundred), whereas some are good at representing very large number of nodes (tens of thousands). Both orthogonal and hierarchical graph drawing require a very clear idea of the graph in terms of the relationships among the nodes before laying out the graph. We use a minimum spanning tree to represent our index and matrix data. A minimum spanning tree is constructed by iteratively discarding the unneccessary edges from a fully connected graph. Therefore, in a minimum spanning tree, no node can be designated as the root. As a result, we cannot use any of the above tree layout algorithms without modifying them to fit our requirements. A computationally expensive solution to this problem is to find a node which is the center of the tree, make that node the root node, and rank all other nodes based on that assumption. However, since finding the root node and computing ranks of nodes in a tree is a problem of significantly high time complexity ($O(n^2)$, where n is the number of nodes), a good compromise would be to consider a random node as the root of the minimum spanning tree, and to lay out the tree starting from that root. This is the technique that we have used in our visualization. Since we have meaningful weights associated with the edges of the amino acid graphs, we want the visualization to be reflective of the distance between two nodes in the tree through the lengths of the edges. To accomplish this, we have used spring embedding on the tree layout algorithm. The spring embedding algorithm adjusts the nodes of the tree after the layout, so that the final layout is a combination of the repulsive forces between the adjacent nodes of the tree and the spring tension forces due to the edges. The spring embedding makes the graph taut and also modifies edge lengths to represent the relationship between adjacent nodes.

3.2 Biovisualization

Visualization of biological data has been an important topic of research in the last decade. There are a number of application areas in biology where visualization plays a key role, including sequence analysis, comparison, and alignment, molecular evolution, genomics, gene expression and gene regulatory networks analysis, and heterogeneous biological database analysis.

Our work is in the area of visualization of heterogeneous biological databases. We are working with the amino acid database, which does not provide a challenge in terms of the size of the database, but calls for new visualization techniques suited for exploring the relationships among different entities belonging to this database. The aim of this thesis is to display 20-dimensional amino acid indices in a spanning tree structure, showing as much detail as possible in a two-dimensional space. We also compare two spanning trees with different numbers of nodes and relate the scoring matrices to the indices to gain insight into the data.

Chapter 4

Minimum Spanning Tree Representation

One of the primary goals of this research is to develop an efficient visualization technique for displaying the indices and the mutation matrices such that the inherent relationships that exist between them are preserved. A connected graph seems to be an obvious choice for representing this relational data model. The original work on which this research is based used a single-linkage hierarchical clustering on the data to understand the relationship between the indices and matrices. The result of hierarchical clustering is a dendrogram, which becomes cluttered when the number of nodes is high. To overcome this problem, Tomii et al. [43] used a minimum spanning tree for visualization; we adopt the same minimum spanning tree representation. A spanning tree of a graph G with n nodes is a connected subgraph G' of G that includes all n nodes and only as many edges as required to make G' a tree (that is $n-1$ edges). A graph can have many spanning trees; a minimum spanning tree (MST) is one in which the sum of the weights of the edges in the spanning tree G' is minimized. In this research, we have two separate views for displaying both the index and the matrix databases. The next section describes the approach for generating these views in details.

4.1 Spanning Tree Generation

We represent the data entities (indices or matrices) as nodes in the spanning tree. The edges are connections between these nodes. To generate a minimum spanning tree from a general graph, a weight (numerical value) needs to be associated with each edge. Following Tomii and Kanehisa we define the weight w for an edge between a pair of nodes as

$$w = 1 - |c|$$

where c is the product moment correlation coefficient of the node vectors, defined as the ratio of the covariance of the two variables to the product of the individual standard deviations [43]. The value of c lies in the range $[-1, 1]$, so w will be in the range $[0, 1]$. Now, for a set of n selected nodes, first the fully connected graph having $n(n-1)/2$ undirected edges is constructed. Every edge in this graph has a weight which is computed using the above equation depending on the values of the nodes connected by that edge. Using these weights, we generate a minimum spanning tree following Kruskal's algorithm [11].

4.2 Spanning Tree Layout

The original work of Tomii et al. [43] does not specify the graph layout scheme used for their spanning tree representation. However, the web-based interactive tool built by Blazej uses the graph layout package provided by Touchgraph, LLC [44]. The naïve way of laying out a MST in screen space is to randomly position the nodes in the space and draw the relevant edge connections. However, a random layout is not acceptable because it does not give us any information about closely related nodes (edges with low weights) and results in unnecessary edge crossings. Figure 4.1 shows an example of a random layout of a MST of forty amino acid indices. It can be seen that the layout is not of much help in getting an intuition about the relationship between the nodes.

To overcome this problem, we use a modified Reingold-Tilford tree layout algorithm [37] for laying out the MST. As discussed in Chapter 3, the Reingold-Tilford algorithm is useful for laying out an ordered tree which has a distinct hierarchical structure (designated root node). However, an MST is a free tree, so the Reingold-Tilford algorithm needs modification before it can be applied to our system.

In order to apply Reingold-Tilford algorithm, we need to designate one of the nodes of the MST as the root. There are three ways we can select the root of the MST: (1) designate a node randomly as the root, (2) designate the node with the minimum of the maximum distances from the leaf nodes as the root, or (3) make the node with the maximum degree the root. The first method does not yield a good layout since there tend to be edge crossings. The second technique has a high running time of $O(n^2)$, where n is the number of nodes.

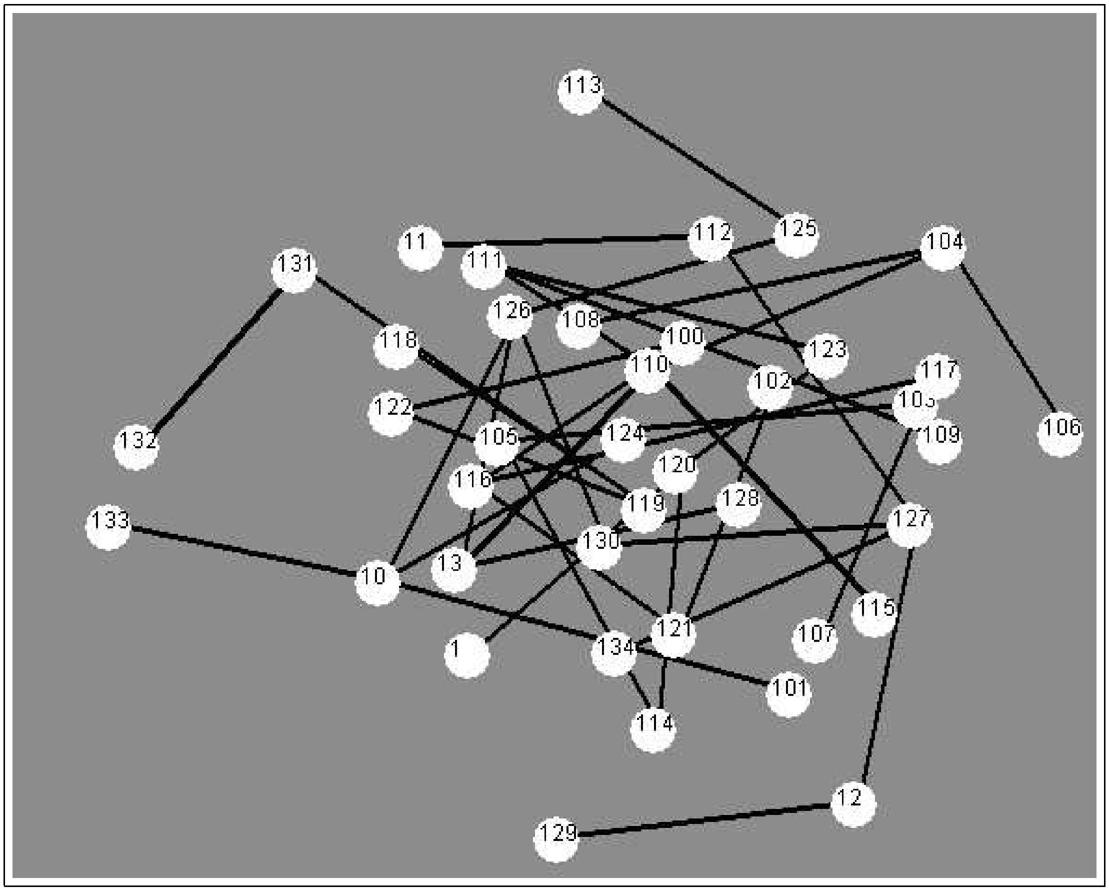


Figure 4.1: Random layout of the MST with 40 indices

In order to avoid such intensive computation, we use the last method for selecting the root of the MST.

After selecting the root node, we assign levels to all nodes in the tree in compliance with their distances from the root node and then apply a modified Reingold-Tilford algorithm. The original Reingold-Tilford algorithm uses only a 180^o space for each node and lays out the tree in a strictly top-down fashion. This leads to wastage of screen space and unnecessarily clutters the layout in the lower levels towards the leaf nodes, since the MST is not a symmetric tree. We modified the algorithm such that each node can use the entire 360^o around it to place a child. This leads to better screen space utilization and less congestion of

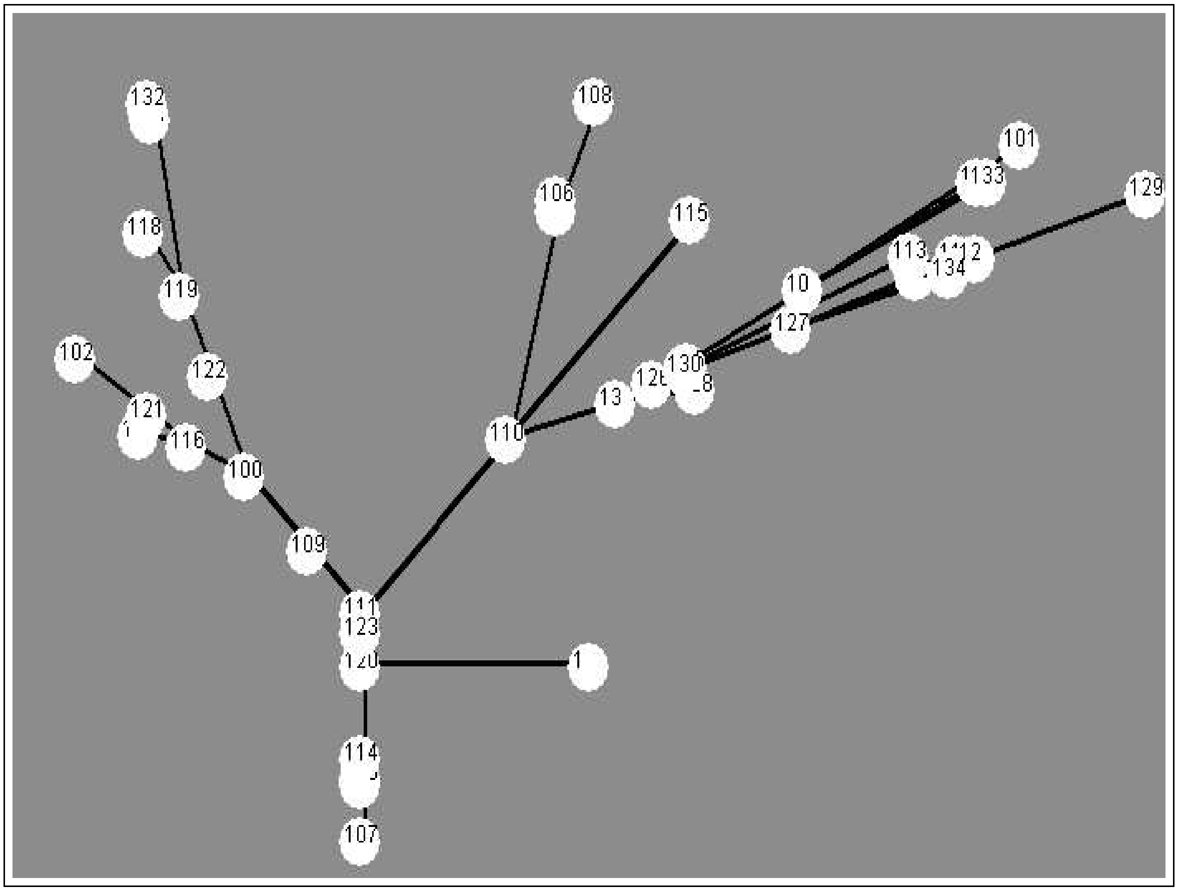
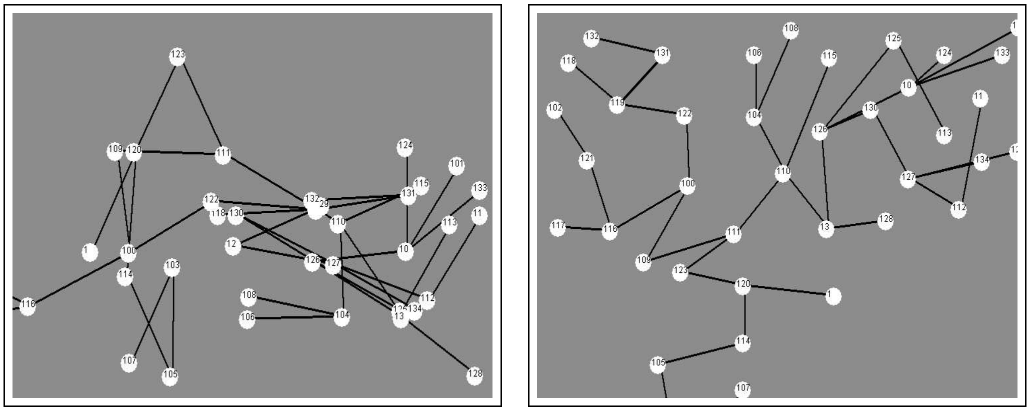


Figure 4.2: Modified Reingold-Tilford layout of the MST with 40 indices

nodes in one area. Also, instead of having fixed lengths of the edges for the tree, we make the edge length proportional to the weights of the individual edges. Figure 4.2 illustrates the layout using our modified Reingold-Tilford algorithm. However, we see in the figure that some nodes are placed almost on top of other nodes. This is because the edge lengths are proportional to the weights of the edges; if these weights are very low, then it results in very short edges.

The solution to this problem is to apply spring embedding after laying out the graph using the modified Reingold-Tilford algorithm. In spring embedding, given a set of nodes and edges between them, the task is to find a configuration of the system such that the total



(a)

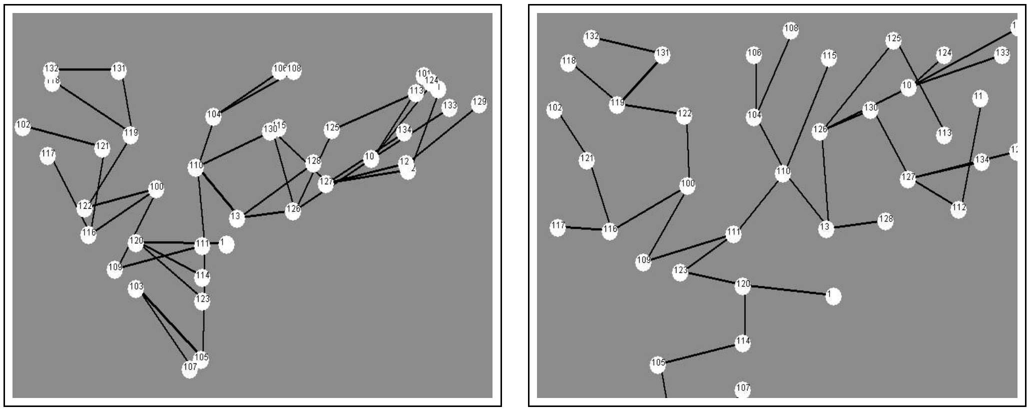
(b)

Figure 4.3: Variation of spring constant (a) $ks = 500$ (b) $ks = 5$

energy of the system is minimized. As a result, the graph is very taut and stable and it has a compact layout. Algorithm 1 describes the spring embedding algorithm [13]. Spring embedding uses two forces to calculate the lengths of the edges: the spring tension force of the edges and the repulsive force of the adjacent nodes. There are two constants in the algorithm; the spring constant (*ks*) and the repulsive force constant (*kr*). The former is set to 5, while we use a value of 150000 for the repulsive force constant. A high value of *ks* (500) makes the final edges long, resulting in edge crossings, as shown in Figure 4.3 **(a)**. When we make *ks* relatively small (5), the layout becomes aesthetically better, as can be seen in Figure 4.3 **(b)**. Similarly, Figure 4.4 **(a)** shows that a small value of *kr* places the nodes on top of each other, whereas increasing *kr* to a value as high as 150000 places the nodes far away from each other, thus preventing obscuration due to partial or complete overlap. Figure 4.4 **(b)** shows the graph layout with *kr*=150000.

Algorithm 1 Spring Embedding

```
$kRepulsiveForce = 50000$
$kSpringConstant = 5$
$edgeLength = 400$
$factor = 0.001$
  for all nodes in the tree do
    pick the first vertex (v)

    //spring tension forces are only because of the adjacent edges, which try to
    //keep the edge length 'lSpringLength'

    for all adjacent edges do
      select the adjacent node (u)
      find the screen distance (only distance) between u and v
      lSpringLength = const * weight of edge between u and v
      forcex = kSpringConstant *(distance - lSpringLength)*(v.x-u.x)/distance
      forcey = kSpringConstant *(distance - lSpringLength)*(v.y-u.y)/distance
      sumofforcesx +=forcex;
      sumofforcesy +=forcey;
    end for

    //repulsive forces are because of all nodes in the system

    for all nodes do
      select a node u
      find distance between u and v
      force = -1 * krepulsiveforce / (distance * distance)
      forcex = force * (v.x - u.x) / (distance)
      forcey = force * (v.y - u.y) / (distance)
      sumofforcesx +=forcex
      sumofforcesy +=forcey
    end for

    v.x = sumforcesx*factor;
    v.y = sumforcesy*factor;

  end for
```



(a) **(b)**

Figure 4.4: Variation of repulsive force constant (a) $kr = 50$ (b) $kr = 150000$

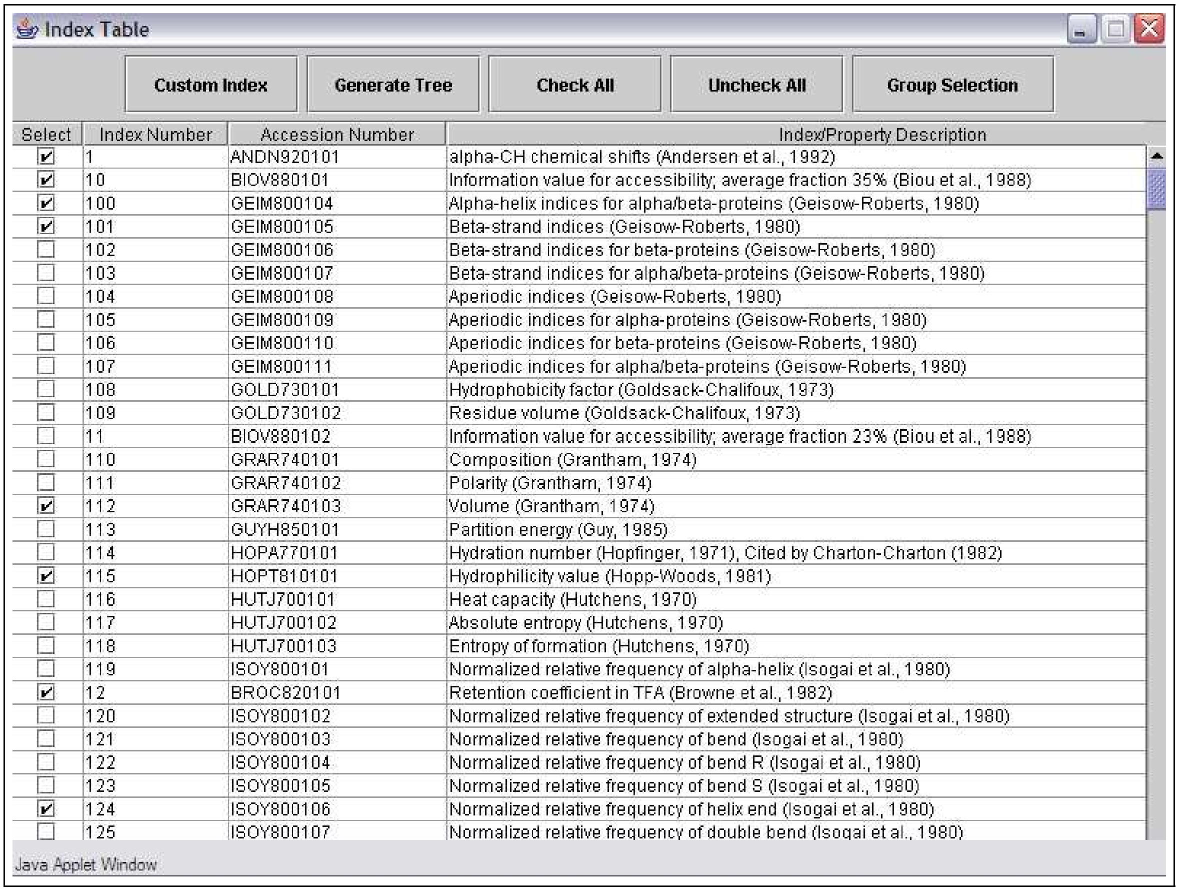
4.3 Index Spanning Tree

The correlation coefficient c for indices is computed as follows [8]:

$$c = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\left[\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2\right]^{1/2}}$$

For any two amino acid indices **X** and **Y**, x_i represents the elements of **X**, and y_i represents those of **Y**. The means of the elements of **X** and **Y** are denoted by $\bar{x}$ and $\bar{y}$, respectively, and the number of elements by n. Since the correlation coefficient c is a measure of the relatedness or dependency of two variables, its difference from 1 can be considered to be the unrelatedness or distance between the variables.

For generating an index spanning tree, the user is allowed to choose a number of indices from a table displaying the 494 available indices from the UMBC AAIndex database(Figure 4.5). Kruskal's algorithm is applied to the fully connected graph constructed from these



Select	Index Number	Accession Number	Index/Property Description
☑	1	ANDN920101	alpha-CH chemical shifts (Andersen et al., 1992)
☑	10	BIOV880101	Information value for accessibility; average fraction 35% (Biou et al., 1988)
☑	100	GEIM800104	Alpha-helix indices for alpha/beta-proteins (Geisow-Roberts, 1980)
☑	101	GEIM800105	Beta-strand indices (Geisow-Roberts, 1980)
☐	102	GEIM800106	Beta-strand indices for beta-proteins (Geisow-Roberts, 1980)
☐	103	GEIM800107	Beta-strand indices for alpha/beta-proteins (Geisow-Roberts, 1980)
☐	104	GEIM800108	Aperiodic indices (Geisow-Roberts, 1980)
☐	105	GEIM800109	Aperiodic indices for alpha-proteins (Geisow-Roberts, 1980)
☐	106	GEIM800110	Aperiodic indices for beta-proteins (Geisow-Roberts, 1980)
☐	107	GEIM800111	Aperiodic indices for alpha/beta-proteins (Geisow-Roberts, 1980)
☐	108	GOLD730101	Hydrophobicity factor (Goldsack-Chalifoux, 1973)
☐	109	GOLD730102	Residue volume (Goldsack-Chalifoux, 1973)
☐	11	BIOV880102	Information value for accessibility; average fraction 23% (Biou et al., 1988)
☐	110	GRAR740101	Composition (Grantham, 1974)
☐	111	GRAR740102	Polarity (Grantham, 1974)
☑	112	GRAR740103	Volume (Grantham, 1974)
☐	113	GUYH850101	Partition energy (Guy, 1985)
☐	114	HOPA770101	Hydration number (Hopfinger, 1971), Cited by Charton-Charton (1982)
☑	115	HOPT810101	Hydrophilicity value (Hopp-Woods, 1981)
☐	116	HUTJ700101	Heat capacity (Hutchens, 1970)
☐	117	HUTJ700102	Absolute entropy (Hutchens, 1970)
☐	118	HUTJ700103	Entropy of formation (Hutchens, 1970)
☐	119	ISOY800101	Normalized relative frequency of alpha-helix (Isogai et al., 1980)
☑	12	BROC820101	Retention coefficient in TFA (Browne et al., 1982)
☐	120	ISOY800102	Normalized relative frequency of extended structure (Isogai et al., 1980)
☐	121	ISOY800103	Normalized relative frequency of bend (Isogai et al., 1980)
☐	122	ISOY800104	Normalized relative frequency of bend R (Isogai et al., 1980)
☐	123	ISOY800105	Normalized relative frequency of bend S (Isogai et al., 1980)
☑	124	ISOY800106	Normalized relative frequency of helix end (Isogai et al., 1980)
☐	125	ISOY800107	Normalized relative frequency of double bend (Isogai et al., 1980)

Figure 4.5: Graphical user interface for displaying the database of amino acid indices. It gives the user the option to select individual indices for display

nodes and the modified Reingold-Tilford algorithm, along with spring embedding, then produces the final layout.

A typical spanning tree view for forty amino acid indices is shown in Figure 4.6. There is an edge between indices 1 and 120, which means that the distance between 1 and 120 is less than the distance from index 1 to any other node. This tells the user that the normalized relative frequency of extended structure (index 120) of an amino acid tends to be directly related to its alpha-ch chemical shifts (index1).

If there is more than one index adjacent to a particular index, the user might want to know

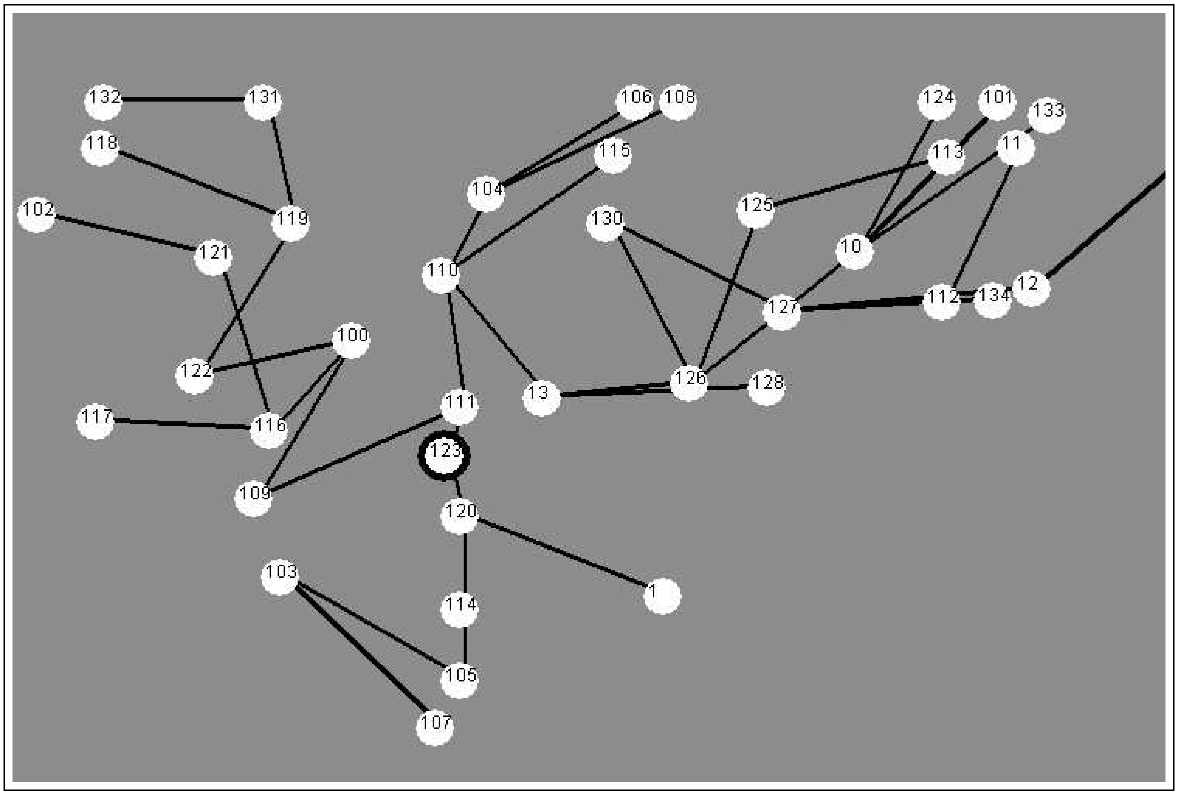
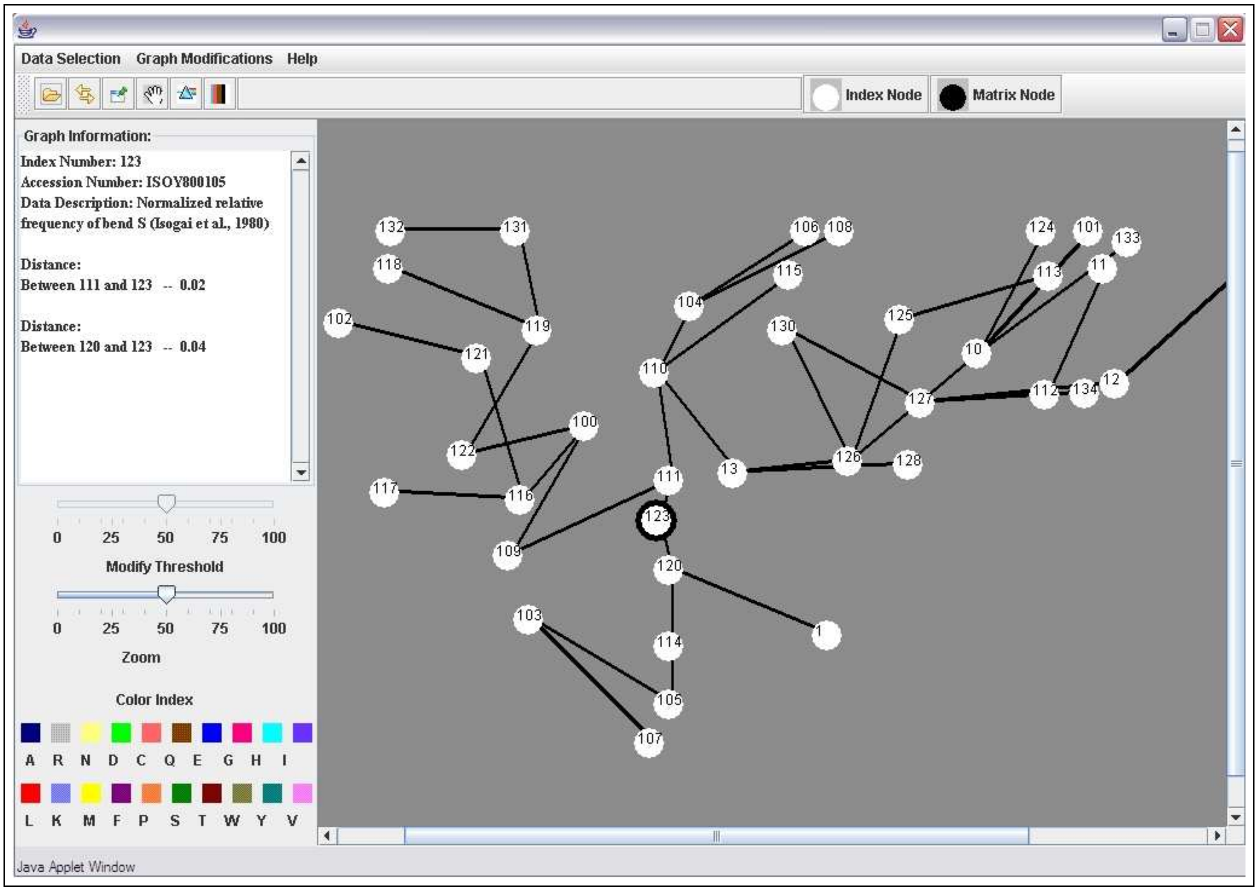


Figure 4.6: Minimum spanning tree from 40 selected indices

which of these indices is closest. Therefore, we print the distance of a selected (clicked) index from all its adjacent indices. In Figure 4.7, we show the distance of index 123 from its adjacent indices, 111 and 120. We can observe that 123 is closer to the two indices 111 and 120 than other indices, such as 1 and 109. However, of these two closest indices, node 111 is closer to node 123, with a distance value of only 0.02. Another important observation is that the spanning tree constructed depends on the nodes that are selected. As an example, index **X** and index **Y** may both be present in two selections of nodes to be displayed. However, there can be an edge between them in the first MST, and no edge between them in the second MST, if there is a different index **Z** that is closer than **Y**to **X** in the second MST.



Index Number: 123
Accession Number: ISOY800105
Data Description: Normalized relative frequency of bend S (Isogai et al., 1980)

Distance:
Between 111 and 123 -- 0.02

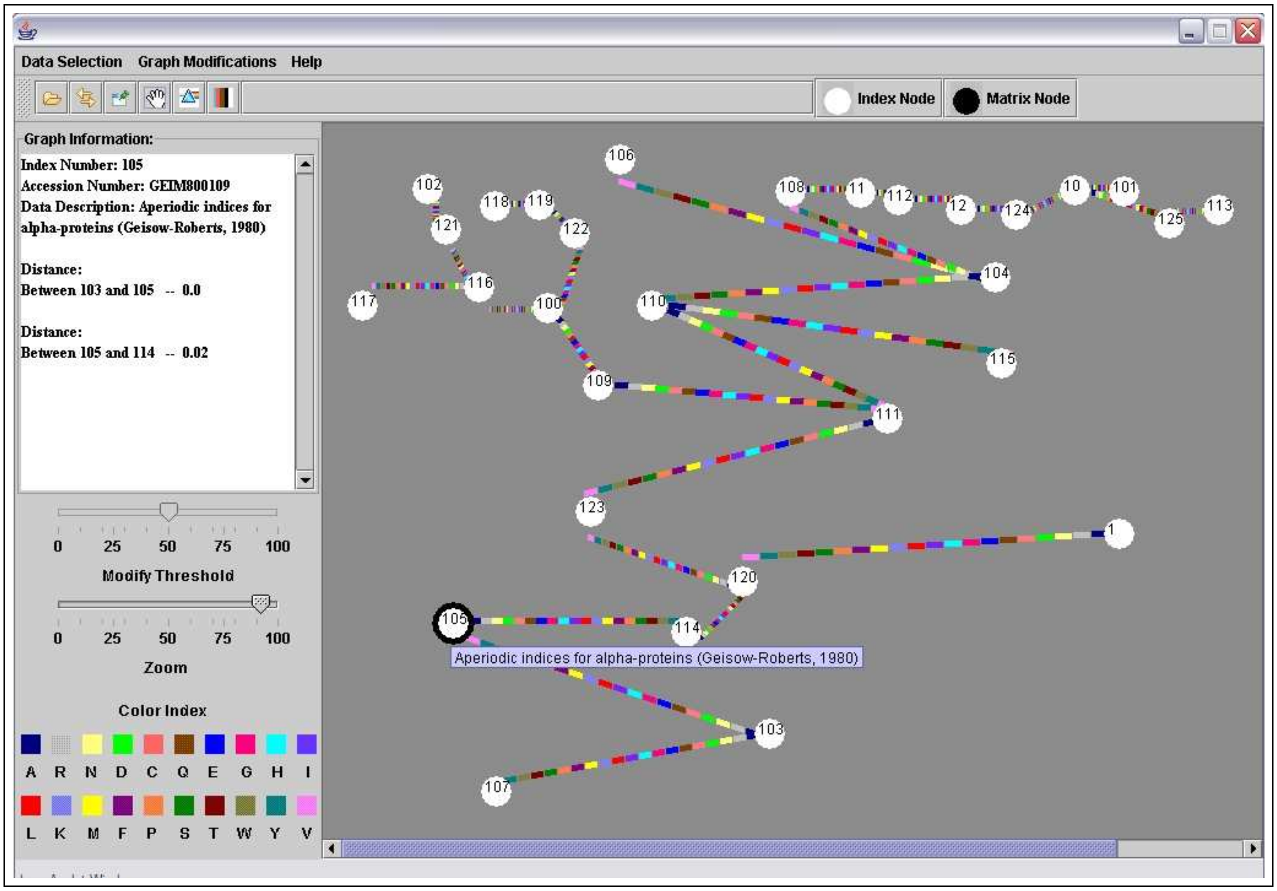
Distance:
Between 120 and 123 -- 0.04

Figure 4.7: Details of Edge Lengths Shown in Text Area for the Index Spanning Tree

4.3.1 Banded Edges

In this spanning tree view of amino acid indices, by studying the adjacency of different nodes, we can draw conclusions about the relatedness of pairs of indices. However, the indices are 20-dimensional vectors, where each dimension is an amino acid. We say that two properties or indices agree with each other when all 20 amino acids have correlated values for the two properties. However, we are not able to deduce anything about the difference of values for a particular amino acid in these two properties. To overcome this difficulty, we enhance the index spanning tree view by incorporating the idea of banded edges. If the user wants to see the difference in values of an amino acid in any pair of indices in a spanning tree, he can display this information by clicking on the “Show Banded Edges” button in the toolbar. In this view, the monochromatic edges of the spanning tree are replaced by thicker lines that are divided into 20 equal parts, each part representing an amino acid. The 20 parts are color-coded, where each color stands for one of the twenty amino acids. The color index is provided in the lower area of the left panel, as shown in Figure 4.8. For example, we can see from the color index in Figure 4.8 that amino acid Alanine (**A**) is dark blue in color. The colors are always arranged in order, starting from the node with the lower index number. For example, in Figure 4.9, the order for the edge between nodes 1 and 120 is from node 1 to node 120, whereas for the edge between nodes 120 and 114 it is from node 114 to node 120.

Once we have a way of representing the 20 amino acids in the graph, we can show how the amino acids differ in the two properties by drawing perpendicular lines through each of the colored blocks. The length of each perpendicular line is proportional to the difference in values of each amino acid for the two properties at the ends of the edge under consideration. However, representing all the perpendicular lines for all edges will render the visualization scheme useless by adding too much clutter or unwanted details. Therefore, the perpendicular lines are displayed for only those edges which are adjacent to a selected (clicked) node. In Figure 4.9, node 109 is selected; the three edges adjacent to index 109 are those connecting it to index 1, index 100, and index 103. The bars are visible for only those three banded edges. The vertical line through the red band is long in the edge between 109 and 1, whereas there is a much shorter bar for the red band in the edge between 109 and 100. This means that the value of amino acid Leucine (**L**) is similar for indices 109 and 100,



Index Number: 105
Accession Number: GEIM800109
Data Description: Aperiodic indices for alpha-proteins (Geisow-Roberts, 1980)

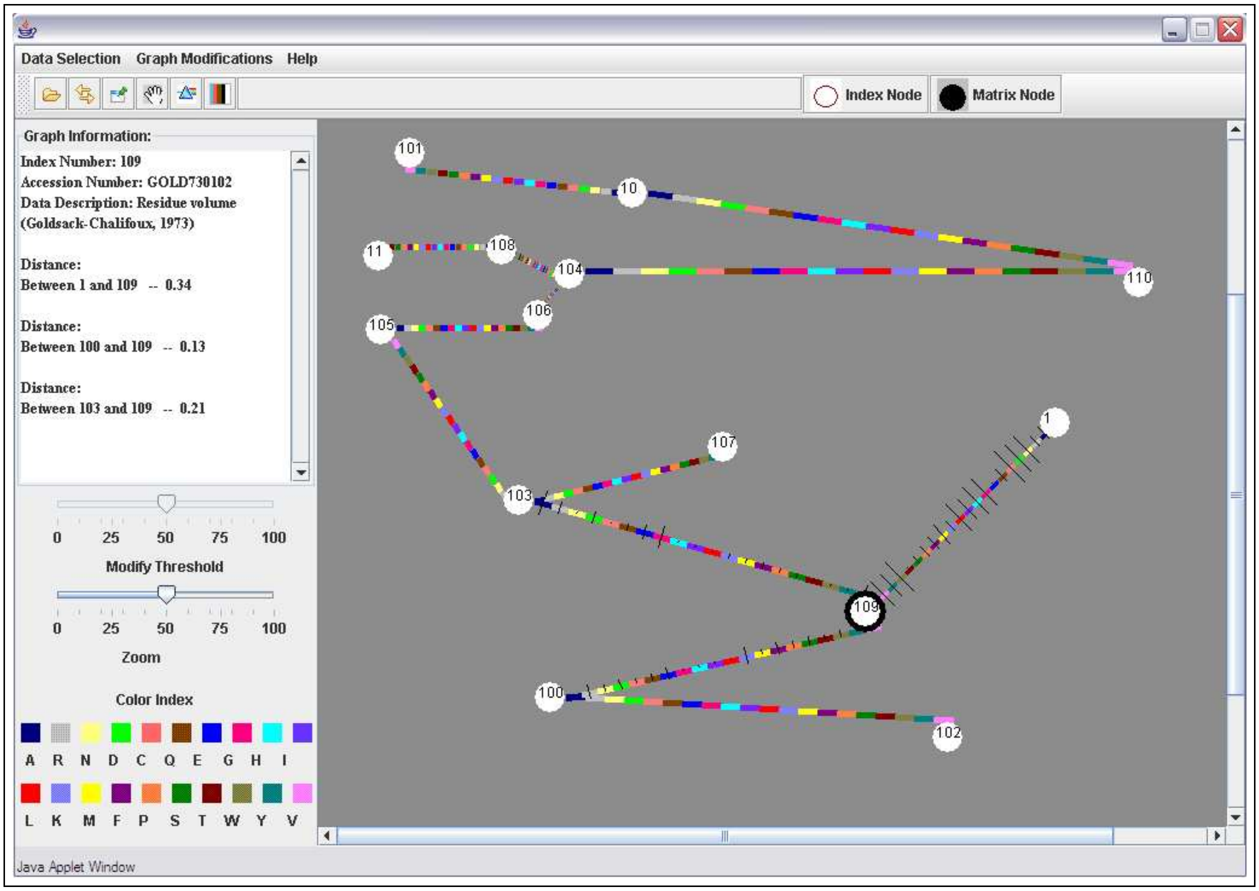
Distance:
Between 103 and 105 -- 0.0

Distance:
Between 105 and 114 -- 0.02

Aperiodic indices for alpha-proteins (Geisow-Roberts, 1980)

Figure 4.8: Banded edges in index spanning tree and color index as part of the graphical user interface

whereas it is very different in indices 109 and 1. In the banded edges view, the absence of a perpendicular line for any amino acid block indicates that the two properties have the same value for both amino acids. On the other hand, the longest length of the line would indicate the maximum difference. Thus, by looking at these three edges in Figure 4.9, we can conclude that the distance between nodes 109 and 1 is much more than the distance between 109 and 100 or between 109 and 103, since the perpendicular bars corresponding to the amino acids in the edge between 109 and 1 are much longer than those for the edges 109-100 or 109-103. The text area to the left corroborates our claim. We can turn on the perpendicular lines on the edges adjacent to a node by clicking on the node. By clicking on



Index Number: 109
Accession Number: GOLD730102
Data Description: Residue volume
(Goldsack-Chalifoux, 1973)

Distance:
Between 1 and 109 -- 0.34

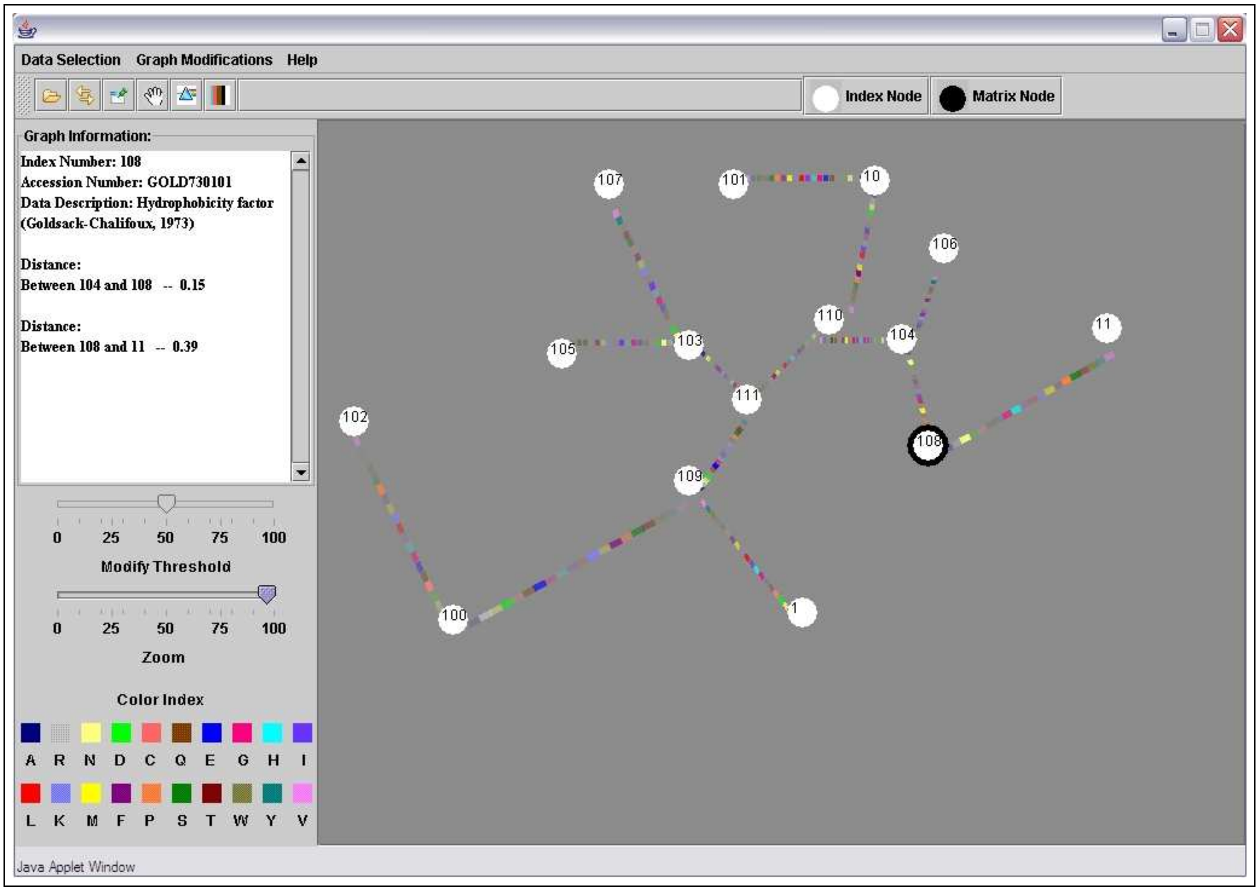
Distance:
Between 100 and 109 -- 0.13

Distance:
Between 103 and 109 -- 0.21

Figure 4.9: Banded edges in the index spanning tree with differences of values of amino acids between the two associated properties shown as bars perpendicular to the blocks. The length of the perpendicular bars is proportional to the difference value.

the same node again, we can turn off the line drawing. Therefore, at any instance, we can click more than one node to get the details of the amino acids for each node. We can again turn off the lines one by one by clicking on any of the selected nodes.

In the method described above, the user needs to click on a node to see the difference in amino acid values between the two indices. But when the system has many nodes, then this method will not be very efficient in representing the individual amino acid information for all the nodes in the tree. To avoid this problem, our system also provides a second way to visualize the difference in amino acid values between any two nodes currently in



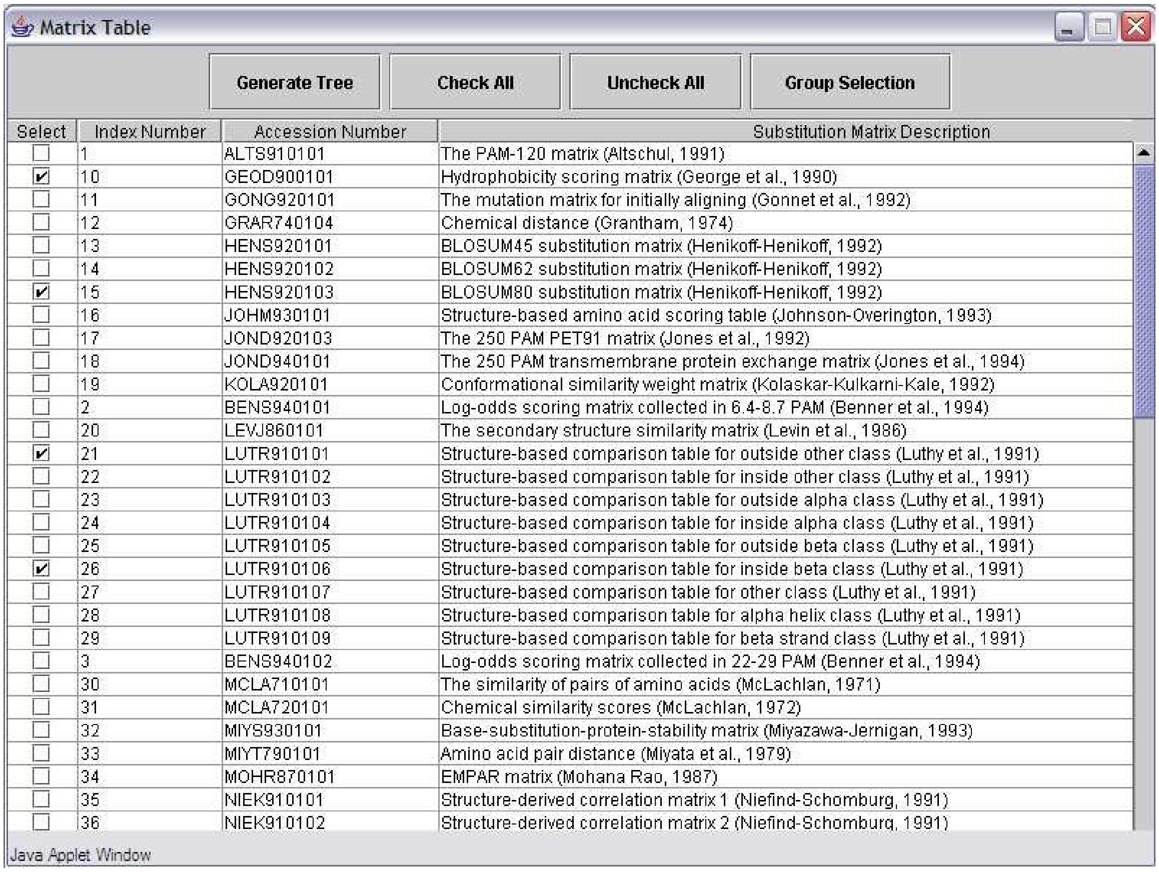
Index Number: 108
Accession Number: GOLD730101
Data Description: Hydrophobicity factor (Goldsack-Chalifoux, 1973)

Distance:
Between 104 and 108 -- 0.15

Distance:
Between 108 and 11 -- 0.39

Figure 4.10: Banded Edges in Index Spanning Tree with α-value mapping of color

the selection. We map the difference in amino acid values between the nodes to the α-value (opacity) of the colors assigned to each of the amino acids. For any two chosen indices, the α-value of the maximum difference of the amino acid values is assigned a value of 1.0 while the minimum difference is assigned a value of 0.2. The rest are linearly interpolated in between. Figure 4.10 shows such a mapping for a set of indices. Among the different amino acids for the edge between nodes 11 and 108, we see that three colors orange, red and cyan are the most prominent, indicating that the difference in amino acid values corresponding to these three colors (namely **N**, **G** and **P**) are very high for the two nodes in consideration.



Select	Index Number	Accession Number	Substitution Matrix Description
☐	1	ALTS910101	The PAM-120 matrix (Altschul, 1991)
☑	10	GEOD900101	Hydrophobicity scoring matrix (George et al., 1990)
☐	11	GONG920101	The mutation matrix for initially aligning (Gonnet et al., 1992)
☐	12	GRAR740104	Chemical distance (Grantham, 1974)
☐	13	HENS920101	BLOSUM45 substitution matrix (Henikoff-Henikoff, 1992)
☐	14	HENS920102	BLOSUM62 substitution matrix (Henikoff-Henikoff, 1992)
☑	15	HENS920103	BLOSUM80 substitution matrix (Henikoff-Henikoff, 1992)
☐	16	JOHM930101	Structure-based amino acid scoring table (Johnson-Overington, 1993)
☐	17	JOND920103	The 250 PAM PET91 matrix (Jones et al., 1992)
☐	18	JOND940101	The 250 PAM transmembrane protein exchange matrix (Jones et al., 1994)
☐	19	KOLA920101	Conformational similarity weight matrix (Kolaskar-Kulkarni-Kale, 1992)
☐	2	BENS940101	Log-odds scoring matrix collected in 6.4-8.7 PAM (Benner et al., 1994)
☐	20	LEVJ860101	The secondary structure similarity matrix (Levin et al., 1986)
☑	21	LUTR910101	Structure-based comparison table for outside other class (Luthy et al., 1991)
☐	22	LUTR910102	Structure-based comparison table for inside other class (Luthy et al., 1991)
☐	23	LUTR910103	Structure-based comparison table for outside alpha class (Luthy et al., 1991)
☐	24	LUTR910104	Structure-based comparison table for inside alpha class (Luthy et al., 1991)
☐	25	LUTR910105	Structure-based comparison table for outside beta class (Luthy et al., 1991)
☑	26	LUTR910106	Structure-based comparison table for inside beta class (Luthy et al., 1991)
☐	27	LUTR910107	Structure-based comparison table for other class (Luthy et al., 1991)
☐	28	LUTR910108	Structure-based comparison table for alpha helix class (Luthy et al., 1991)
☐	29	LUTR910109	Structure-based comparison table for beta strand class (Luthy et al., 1991)
☐	3	BENS940102	Log-odds scoring matrix collected in 22-29 PAM (Benner et al., 1994)
☐	30	MCLA710101	The similarity of pairs of amino acids (McLachlan, 1971)
☐	31	MCLA720101	Chemical similarity scores (McLachlan, 1972)
☐	32	MIYS930101	Base-substitution-protein-stability matrix (Miyazawa-Jernigan, 1993)
☐	33	MIYT790101	Amino acid pair distance (Miyata et al., 1979)
☐	34	MOHR870101	EMPAR matrix (Mohana Rao, 1987)
☐	35	NIEK910101	Structure-derived correlation matrix 1 (Niefind-Schomburg, 1991)
☐	36	NIEK910102	Structure-derived correlation matrix 2 (Niefind-Schomburg, 1991)

Figure 4.11: Graphical user interface for displaying the database of similarity matrices. It gives the user the option to select individual matrices for display

4.4 Matrix Spanning Tree

In the previous section, we described in detail how an index spanning tree is constructed and displayed. In this section, we describe how the same technique can be applied to construct a spanning tree of the amino acid similarity matrix data. There are 83 similarity matrices. The table user interface for selecting from the 83 available matrices is shown in Figure 4.11. Once the user selects a number of matrices by clicking on the checkboxes, he can generate the tree by clicking on the “Generate Tree” button.

If two matrices are similar, then for any amino acid pair, the similarity value provided by

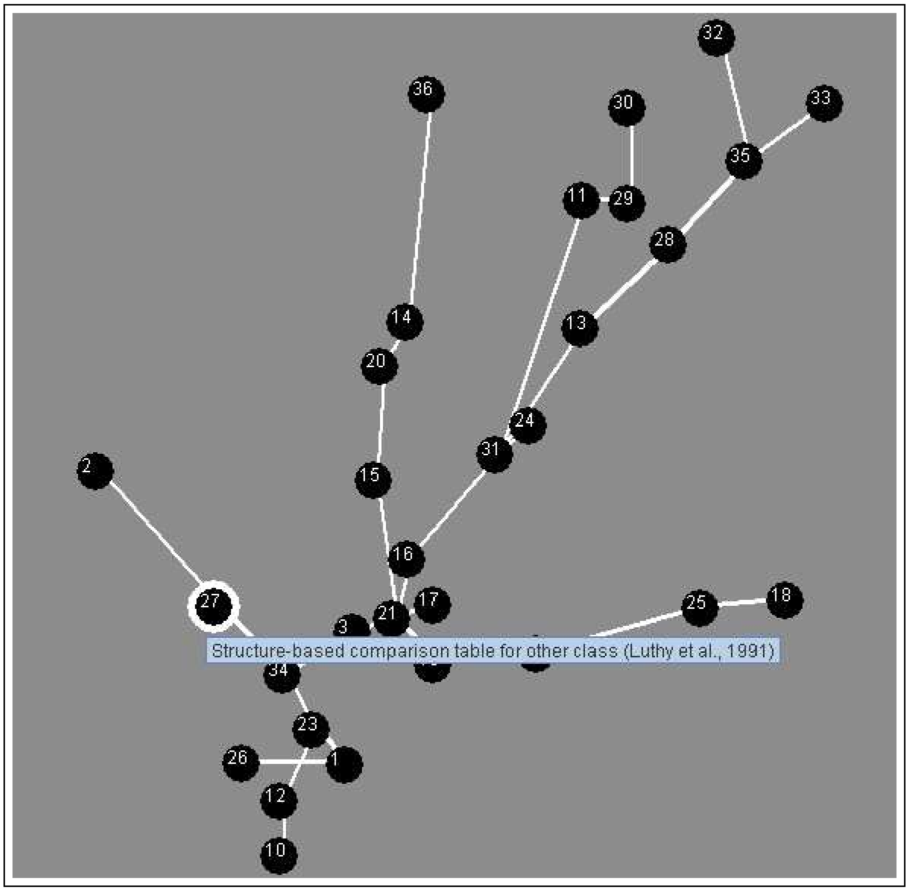
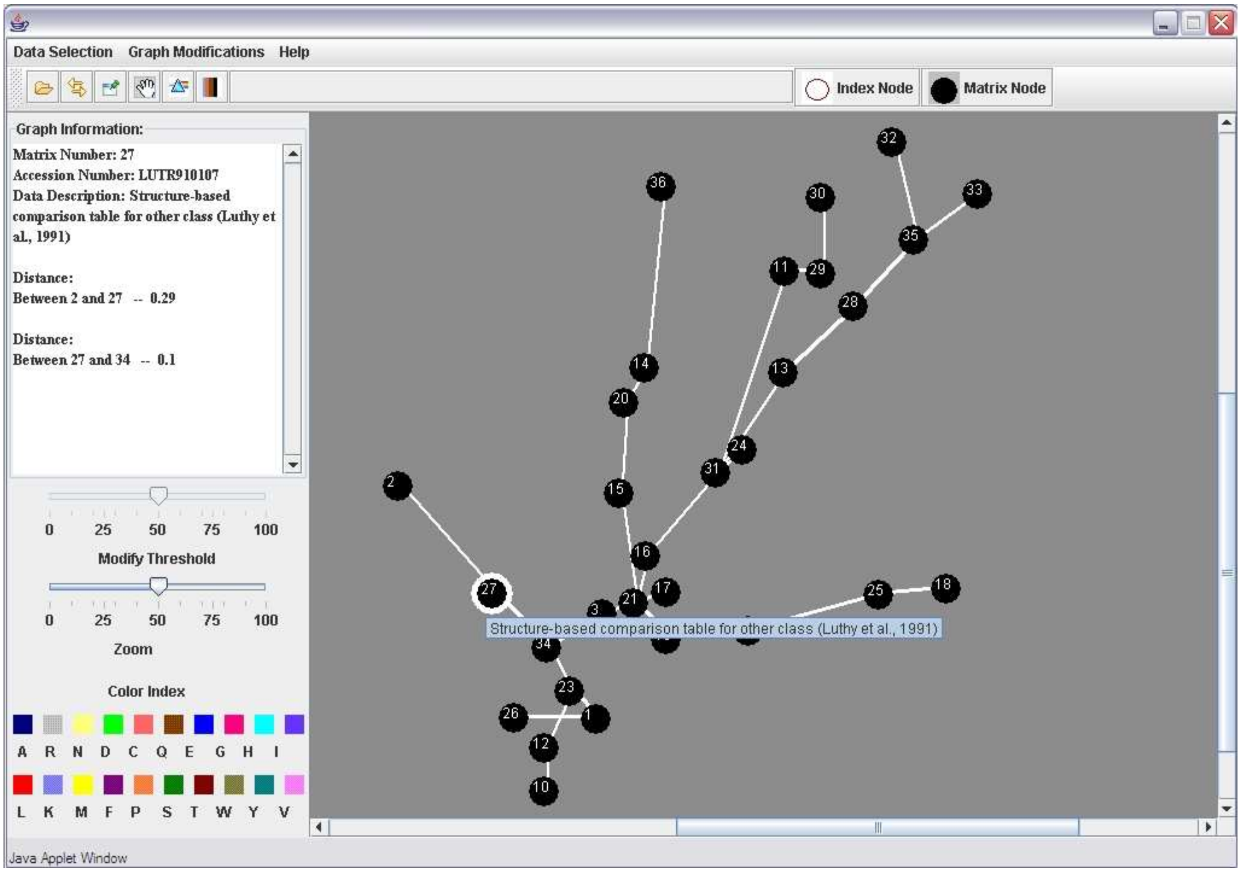

32
36
30
33
35
11
29
28
14
13
20
24
31
2
15
16
27
21
17
25
18
3
Structure-based comparison table for other class (Luthy et al., 1991)
34
23
26
1
12
10

Figure 4.12: Minimum Spanning Tree of 45 Similarity Matrices

one of the matrices is very close to that of the other one. To measure these distances, we use the same distance metric discussed in the previous section. However, we need to define the meaning of the weight metric w with respect to a 20×20 matrix, as opposed to distance between two 20-dimensional column vectors. To compute the distance between two matrices, we transform the 2-dimensional 20×20 data to a 1-dimensional vector of size 400, and then find the distance (weight) between the two 1-dimensional vectors in the same way as the indices (using the correlation coefficient). Given this data, we generate a fully connected graph, from which we generate a MST using Kruskal's algorithm.

Figure 4.12 shows a spanning tree constructed from forty-five selected similarity matrices.



Matrix Number: 27
Accession Number: LUTR910107
Data Description: Structure-based comparison table for other class (Luthy et al., 1991)

Distance:
Between 2 and 27 -- 0.29

Distance:
Between 27 and 34 -- 0.1

Structure-based comparison table for other class (Luthy et al., 1991)

Figure 4.13: Details of Edge Lengths Shown in Text Area for the Matrix Spanning Tree

There is an edge between matrices 2 and 27, which means that the distance between 2 and 27 is less than the distance between 2 and any other matrix. Matrix 2 is the log-odds scoring matrix collected in 6.4-8.7 PAM, whereas matrix 27 is a structure-based comparison table. Although these two similarity matrices were computed under different experimental conditions using different comparison metrics, they agree in predicting which pairs of amino acids are most likely to substitute for one another.

We also provide the user with all information regarding distances between a chosen matrix and all its adjacent matrices. This becomes useful when a matrix has a number of adjacent matrices and the user wants to know which of these adjacent nodes is closest to the concerned matrix, as shown in Figure 4.13. Here, node 27 is connected to nodes 2 and 34. By clicking on node 27, the user can find out the distance of the adjacent nodes from 27 and determine the closest among them (node 34 in this case). Due to the spring embedding of the graph, edge lengths are closely correlated to the weights of the edges.

4.5 Evaluating the Spanning Tree Visualization

We have designed and performed a number of experiments to assess the performance of the layout algorithms in terms of the tasks these graph-based visualizations are intended to support. Given a set of indices or matrices, the spanning tree view lets the user identify the closest node in the tree to any chosen node. To facilitate this task, the lengths of the edges between every pair of nodes should reflect the “closeness” of two nodes. Since the graph is a minimum spanning tree, if a node has only one edge, then that edge obviously connects it to the node to which it is closest. If no edge exists between a pair of nodes, then the user can definitely conclude that those two nodes do not share the minimum distance and their distance in screen space is not considered for analysis. Therefore, we are only concerned with those nodes with more than one adjacent edge. In the ideal case, our visualization should enable the user to judge which of the adjacent nodes is the closest just by looking at the lengths of the edges.

In this section, we report the results of four experiments that we conducted. Throughout the rest of this section we denote the weight of an edge as the computed distance and the length of the edge in screenspace as the physical distance. In the first set of experiments we measure the difference between the computed distance and the physical distance, on average, over all the edges. We call this difference the error; it quantifies the relationship between these two distances. The lower the error, the more effective the visualization. This experiment has been conducted for both the index and the matrix spanning tree. If a node has only one adjacent node, the task of finding its closest node is trivial, since, in the MST constructed, it will be adjacent to only its nearest node. However, if a node in the MST has more than one adjacent node, then the layout of the tree plays a significant role in helping the user to find the closest node.

In the second set of experiments, we evaluate the efficiency of our layout algorithm by measuring the average difference between the lengths of the two edges adjacent to a node, that have the lowest and the second lowest weights. If this difference is high, then, on average, it should be possible to distinguish the closest pairs of nodes in a tree. This experiment has also been done for both the index and the matrix spanning trees. Note that our layout algorithm is deterministic; therefore, given a set of nodes, we will have the exact same layout for different runs of the same experiment.

The first experiment that we have performed on the index spanning tree and the matrix spanning tree is evaluating the total error in the graph. Let w_i be the weight of an edge and d_i be the corresponding physical distance between the nodes connected by that edge. The total error is defined as

$$E = \sum_{i=1}^{n} |\frac{w_i}{W} - \frac{d_i}{D}|$$

where $W = \sum_{i=1}^{n}(w_i)$ and $D = \sum_{i=1}^{n}(d_i)$. This experiment gives us an overall idea about the agreement of these two distances over all the edges of any displayed MST. This error metric can take a minimum value of 0 and its asymptotic upper bound is 1. In the ideal case, the error can have a value of 0, when for each edge, $\frac{w_i}{W} = \frac{d_i}{D}$; that is, when the physical distances are perfectly proportional to their weights. The smaller the value of this metric, the more accurate is the representation. The error will have a value of 0 when we use only the modified Reingold-Tilford algorithm, with no spring embedding, since we make the lengths of the edges proportional to their weights. Because this approach results in a layout where the nodes may be on top of each other, we introduce spring embedding, which increases the error, but improves the aesthetics. Figure 4.14 is the plot of the total error with respect to the number of index nodes in the spanning tree. In this experiment, we select n nodes and lay them out using the tool. Although our layout algorithm is deterministic, we take an average of ten trials by selecting a different set of n indices for each value of n. In Figure 4.14 we have also plotted three baseline cases: the green line represents the layout in which the nodes are laid out using modified Reingold-Tilford and no spring embedding has been applied to the layout; the black and the blue lines show the errors for random layouts. The black line represents the layout where we apply spring embedding after the layout, while the blue line shows the error for random layout without spring embedding.

Not surprisingly, the error is high for the two random layouts, since there is no correspondence between the physical lengths of the edges and their weights. On the other hand, the error is near 0 for the green line, since the edge lengths are perfectly proportional to the weights.

The red line represents the error for our spanning tree visualization. We see that this layout performs much better than the random layout, because although spring embedding repositions the nodes, it preserves the relationships between the weights and the physical distances, thereby keeping the error lower. We also notice that this error decreases as the number of nodes increases. This can be attributed to the effect of the repulsive force of spring embedding. This force is exerted on a node by all other nodes in the graph. When the number of nodes is large, then the repulsive force exerted by some nodes is nullified by that of other nodes, resulting in lower error.

Finally, we note that the error is minimum when we do not use any spring embedding. However, this layout is aesthetically not very pleasant, since at the end of the layout phase, the nodes with low weights are placed almost on top of each other. Therefore, in the spanning tree view, we try to achieve a balance between error minimization and an aesthetically pleasing layout and we optimize the layout algorithm parameters accordingly.

Figure 4.15 shows an error plot for the matrix nodes. Here also the red line represents the error for our matrix spanning tree visualization scheme. The nature of the graph is very similar to that of the index spanning tree.

In the second set of experiments, we have evaluated how efficient our visualization is, in highlighting the edge with the lowest weight, for a node having multiple edges. Our second set of experiments were performed on a subset of the nodes, because the nodes having a single adjacent node have only one edge, which represents the minimum weight. We only consider those nodes in the graph that have more than one adjacent edge. For each such

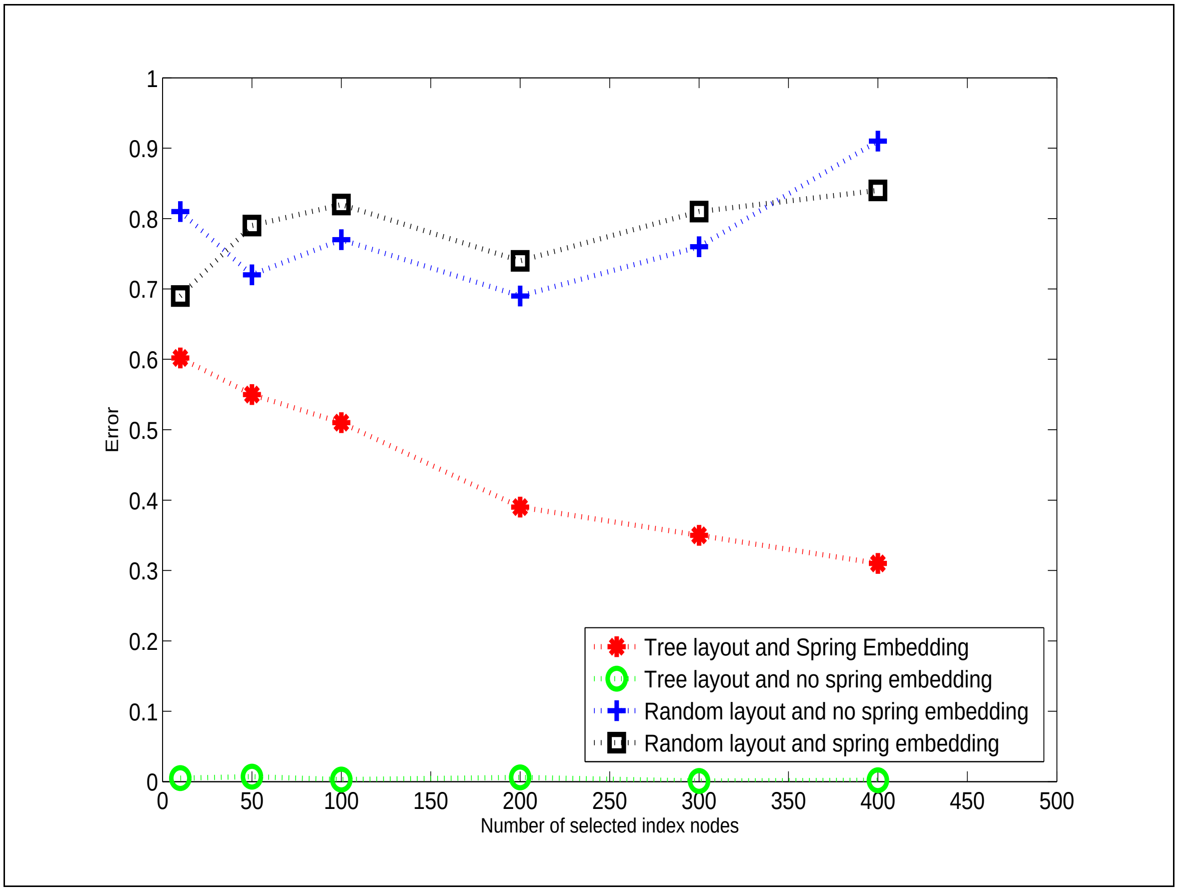


Figure 4.14: Experimental Results Showing the Total Error for Different Sizes of the Index Spanning Tree

node, we find the edges with the least and the second lowest weights. We compute the difference in the physical lengths of the two edges. We repeat this process for all nodes in the graph and take an average over all the differences. We plot these average difference values against *n*, which is the number of selected nodes.

We plot two such graphs for both the index and the matrix spanning tree (Figure 4.16 and Figure 4.17). In Figure 4.16, we see five lines. The blue line shows the results using a tree layout algorithm for constructing the spanning tree, but, no spring embedding. The cyan line corresponds to a random layout for the spanning tree and no spring embedding. The magenta line represents the case where we choose a random layout for the spanning tree

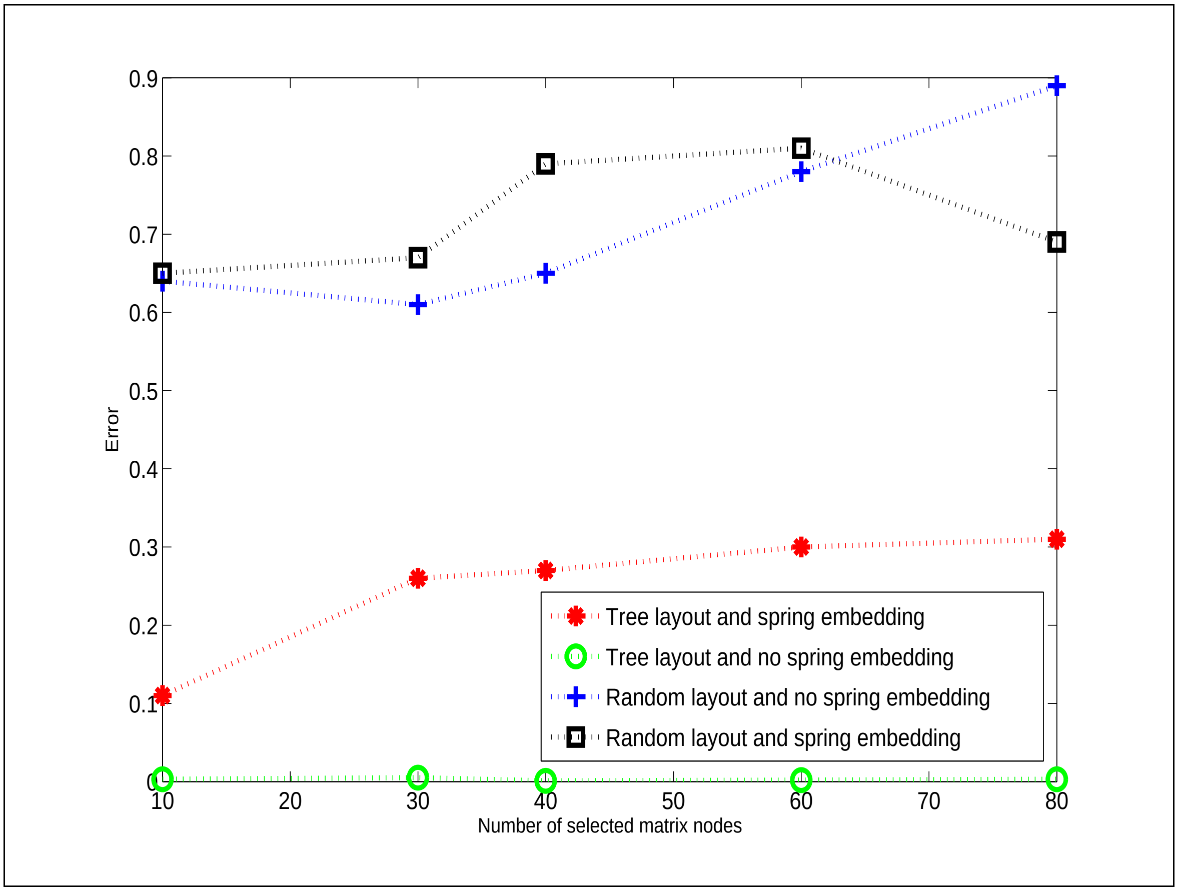


Figure 4.15: Experimental Results Showing the Total Error for Different Sizes of the Matrix Spanning Tree

and display it after spring embedding.

The red and the green lines are the results of the evaluation of our visualization. The red line is plotted after taking an average of the absolute differences, whereas the green line is plotted after taking an average of the exact differences between the minimum and the second minimum edge lengths. In the ideal situation, this difference should always be positive; therefore, the two lines should coincide. If the red and the green lines differ to a great extent, it indicates that the edge having the minimum weight does not have the minimum length in the screen space.

We notice that, except for a couple of observations, the average of the absolute value of the difference is the same as the exact difference. This means that, in most cases, an edge with the lowest weight has the shortest length. With all 494 index nodes selected, the number of reversals of minimum and second minimum edge lengths is never more than 15%, which is evident from the red and the green graphs. The reversal occurs only when a node is dislocated from its correct location due to the combined attractive and repulsive forces as a result of spring embedding. The cyan and magenta lines, representing the difference metric for random layouts with and without spring embedding, have a somewhat arbitrary nature, since the layout does not have any control over placement of the nodes. The blue line has a difference value that is exactly proportional to the difference in the minimum and the second minimum weights. In all the results in Figure 4.16, we see that the average absolute difference is at least twenty-five, which is above the normal human perceptible distance of five pixels [35]. This indicates that our spanning tree visualization successfully depicts which are the two closest nodes in the tree. Figure 4.17 shows very similar observations for the matrix spanning tree.

Our experiments prove that it is always possible for the user to identify the closest node from a group to a given node by looking at the edge lengths. The first metric shows that the overall error increases very slowly with the number of nodes. The system is highly scalable and is easily able to present a large number of nodes. From the second metric, we know that the difference between the minimum and second minimum lengths is greater than the human perceptible range in eighty percent of the cases. Thus we can conclude that our visualization scheme works fairly well in accomplishing what it is designed to do. The error

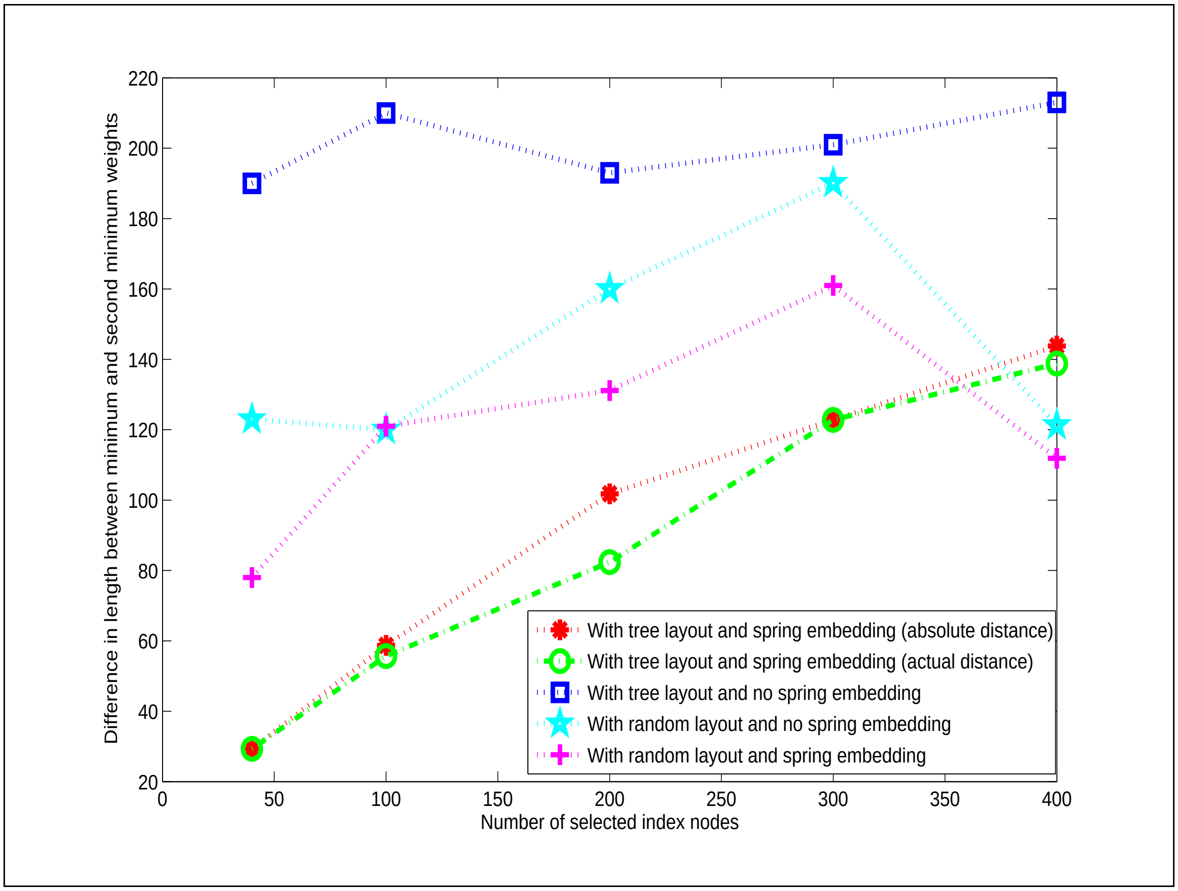


Figure 4.16: Experimental results showing the average difference in lengths of the edges representing the minimum weight and the second lowest weight. The red graph represents the average of the absolute difference whereas the green graph represents that of the exact difference.

graphs tell the users that for the index spanning tree, the graph becomes more accurate with increasing size. Therefore, by looking at a spanning tree, the user should be able to spot the closest pairs of connected nodes. For the matrix spanning tree, the error rate is slightly higher than the index spanning tree. The second graph shows that in approximately 85% of the cases, the user will be able to correctly say which node of a group of adjacent nodes is closest to the concerned node by visually observing the differences in the edge lengths.

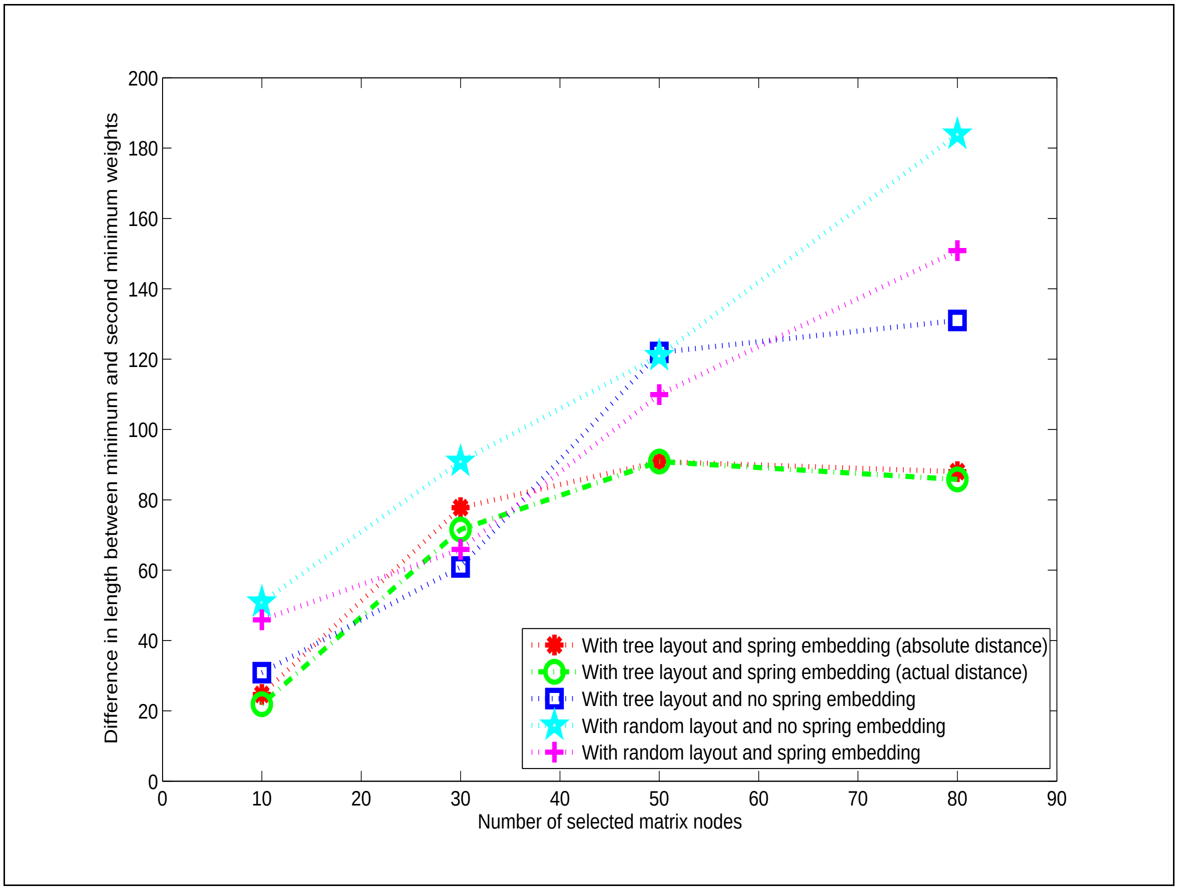


Figure 4.17: Experimental results showing the average difference in lengths of the edges representing the minimum weight and the second lowest weight. The red graph represents the average of the absolute difference whereas the green graph represents that of the exact difference.

Chapter 5

Comparing Spanning Trees

In the previous chapter, we discussed how we use a spanning tree representation to display amino acid indices and the amino acid similarity matrices. The index spanning tree depicts the relationship among different physical and chemical properties of amino acids and how the values of these properties vary from one amino acid to another. The matrix spanning tree shows the relationship among different similarity matrices, where each similarity matrix tells us about the similarity between pairs of amino acids calculated on the basis of empirical results. Given these two spanning trees, an obvious idea would be to link these two trees in some way and deduce some information from the joint structure. The similarity matrices have been generated from experimental results and are, therefore, observed values. Biologists have argued that there must be some pattern in these observed similarities that is affected by the underlying properties of the amino acids [48],[18].

Therefore, we try to compare the two spanning trees by choosing an appropriate layout that will enable a user to infer about the influence of the indices on the matrices.

In the literature of graph visualization, there are a number of ways to compare two graphs. Biologists compare evolutionary trees to come to important conclusions about the theory of evolution. These evolutionary trees may include tens of thousands of nodes. TreeJuxtaposer [32] is a tool for visualizing and comparing such evolutionary trees without loss of focus and context. However, TreeJuxtaposer and other tree visualization tools work under the basic assumptions that the trees have approximately the same number of nodes and that the nodes and edges in the trees represent the same kind of objects and relationships. In our system, we cannot make this assumption. One of the trees is a tree of indices, which are twenty-element vectors, whereas the other one is a tree of matrices, which are 20×20 arrays. Furthermore, the numbers of nodes in the two trees are not necessarily equal.

Several problems can be solved using this visualization. We might want to know how a particular index influences a group of similarity matrices. In that case, we would have to compare one node of the index spanning tree to many nodes of the matrix spanning tree. Alternatively, given a number of indices and a single matrix, we might want to find out which index is most responsible for the values in the given similarity matrix. In this case, we have many nodes of the index spanning tree and only one node of the matrix spanning tree.

5.1 Layout Algorithms

In this chapter, we describe three 2D layout algorithms for jointly laying out two independent spanning trees to enable a user to understand the relationship and influence of one spanning tree on the other. The three algorithms are *joint independent layout*, *joint layout influenced by the index tree*, and *joint layout influenced by the matrix tree*.

The *joint independent layout* approach explores the relationship between the nodes of the two spanning trees without any constraints on the layout of the two trees. In the *joint layout influenced by the index tree*, the layout of the matrix spanning tree is influenced by the positions of the nodes of the index spanning tree. The index tree is first drawn using the standard spanning tree layout algorithm. The nodes of the matrix tree are then positioned near the nodes of the index tree to which they are the closest. In the third layout (*joint layout influenced by the matrix tree*), the matrix tree is first laid out and the index nodes are placed relative to the matrix nodes.

5.1.1 Joint Independent Layout

In this layout, the user first selects a number of indices from the index database and a number of similarity matrices from the similarity matrix database. The first step of the algorithm is to construct the two spanning trees using the technique described in the last chapter. We then compute the distance between a matrix node and an index node for every index-matrix pair. We refer to this distance as the computed inter-tree distance. The indices are normalized values of specific amino acid properties. For comparing one such index to a similarity matrix we need to establish a relationship between these two amino acid data. The difference between the i^{th} and the j^{th} entries of an amino acid index would tell us about the disaggrement (or agreement) between two amino acids for a specific property. Therefore we can create a matrix of these differences and use that to compute the inter-tree distances. Since one of these is a one dimensional vector of size 20, and the other is a two-dimensional vector of size 20×20, we first apply a transformation on both to make their dimensions agree so that mathematical operations can be applied on them. Given an index vector, we first derive a 20×20 matrix where the $(i,j)^{th}$ element of the matrix is the difference between the i^{th} and j^{th} element of the index vector. We then convert this 20×20 matrix to a 400×1 vector. We also convert each 20×20 matrix to a 400×1 vector. Now that we have two vectors, we can easily compute the distance of these two vectors using the distance equation of the previous chapter. Note, however, that this distance metric might not be the best representative of the distance relationship between the indices and the matrices, since the indices are linearly normalized values of the properties, and the matrices are log odds of the original probabilities of the substitution matrices.

The next step of the algorithm is to lay out the two spanning trees as part of a single graph (which may not be a spanning tree). We use spring embedding after laying out this new graph, which has two types of nodes (index and matrix nodes) and three types of edges (index-index, matrix-matrix, and index-matrix edges). We optimize this layout algorithm for effective visualization. Instead of drawing all of the edges between each matrix node and each index node, we display only those index-matrix edges that are greater than a predefined threshold. This reduces visual clutter by reducing the number of index-matrix edges. If we were generating a graph from an index tree with m nodes and a matrix tree with n nodes, then without the threshold, we would have to display $m \times n$ index-matrix edges, which would make the graph very difficult to interpret. In our system, we provide the option of changing the threshold dynamically, so that at any time the user will only see the edges whose distances are below the chosen threshold. To make this visualization more interpretable, we have mapped the opacity of the index-matrix edges based on the distances they represent. For this purpose, we have normalized every index-matrix distance to have a value between 0 and 1. We then map a distance of 0 to an opacity of 1 (completely opaque) and a distance of 1 to an opacity 0 (transparent) and we linearly interpolate the opacities of all the edges.

Figure 5.1 shows the joint independent layout as implemented in our system. We have selected ten index (white) nodes and ten matrix (black) nodes. The red edges are the index-matrix edges. One can easily note the reduction of the number of index-matrix edges from 100 ($m \times n$) to approximately 30, with a threshold of 0.5. Standard functions such as graph moving, node pinning and zooming are available in this view.

5.1.2 Joint Layout Influenced by the Index Tree

In the joint independent layout, the positions of the nodes in the graph do not depend on the index-matrix distances. Therefore, an index node and a matrix node might be placed far away from each other, even if the distance between them is small. This not only leads to

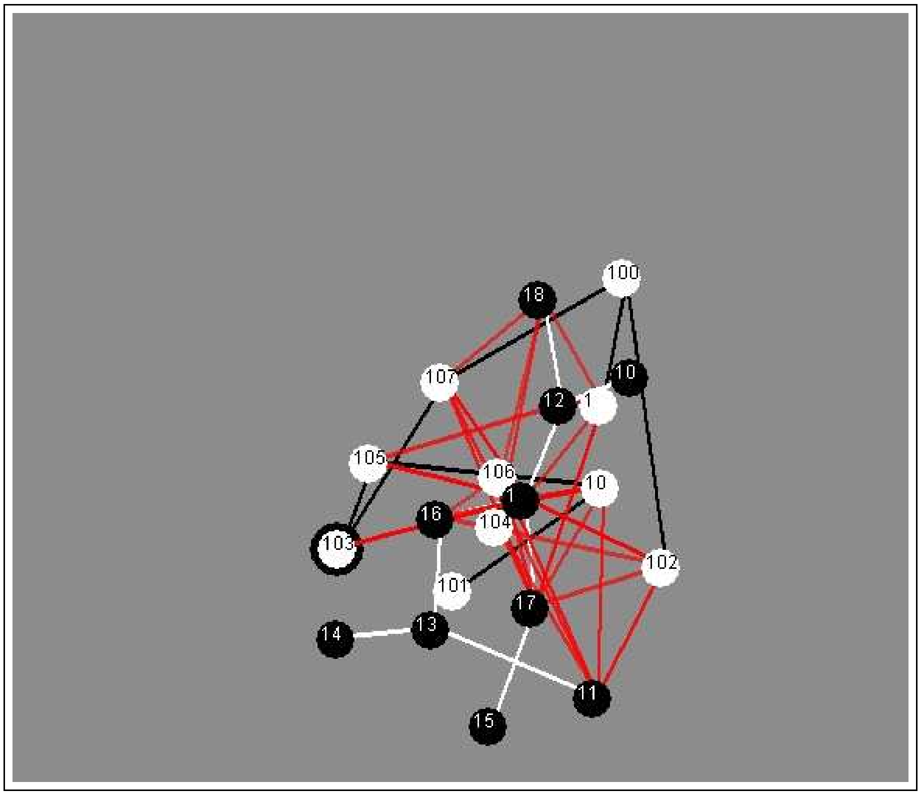


Figure 5.1: Joint independent layout where positions of matrix nodes are determined by spring embedding

unnecessary edge-crossings in the visualization, but also reduces the interpretability of the graph in terms of analyzing the relationships between index and matrix nodes. In the joint layout influenced by the index tree, we solve this problem by determining the position of each matrix node based on the closest index node.

We first select a set of index nodes, lay out the index spanning tree, and then lay out the nodes of the matrix spanning tree after computing the index-matrix distances. After all such distances are computed for each matrix node, we arrange the distances in ascending order. We then choose the edges that are to be drawn based on the current threshold. We place every matrix node closest to the index node with the smallest computed inter-tree distance from it. At the end of the layout process, all the matrix nodes have been placed near their respective closest index nodes.

If an index node is closest to more than one matrix node, and if all of them have the same distance from the index node, then those matrix nodes are placed radially around that index node at an equal physical (screen space distance) distance from that index node. The physical distance is a function of the computed inter-tree distance ($w = 1 - c$) between the index matrix pair.

This layout helps us identify a group of matrices that is close to an index. However, choosing the threshold plays an important role in this visualization. The lower the distance between an index-matrix pair, the more significant the edge. When the threshold is relatively low, the user can visualize only the most significant inter-tree edges. As the threshold increases, the number of such edges also increases. Unlike the joint independent layout, a change in the threshold value might lead to a change in the position of a matrix node, depending on whether one of the edges originally displayed has to be removed or whether a new edge has to be added to the display. The similarity between matrix nodes does not affect this layout at all.

An important aspect of this layout is the study of the resulting matrix tree. In a matrix spanning tree, nodes are adjacent to each other only if the distance between them is minimum. The similarity matrices are based on observations that the biologists are trying to explain using the amino acid index data. If two matrix nodes have an edge between them, then they are very similar to each other; they should, therefore, have very similar amino acid indices influencing them. If in this layout influenced by the index spanning tree, the matrix nodes

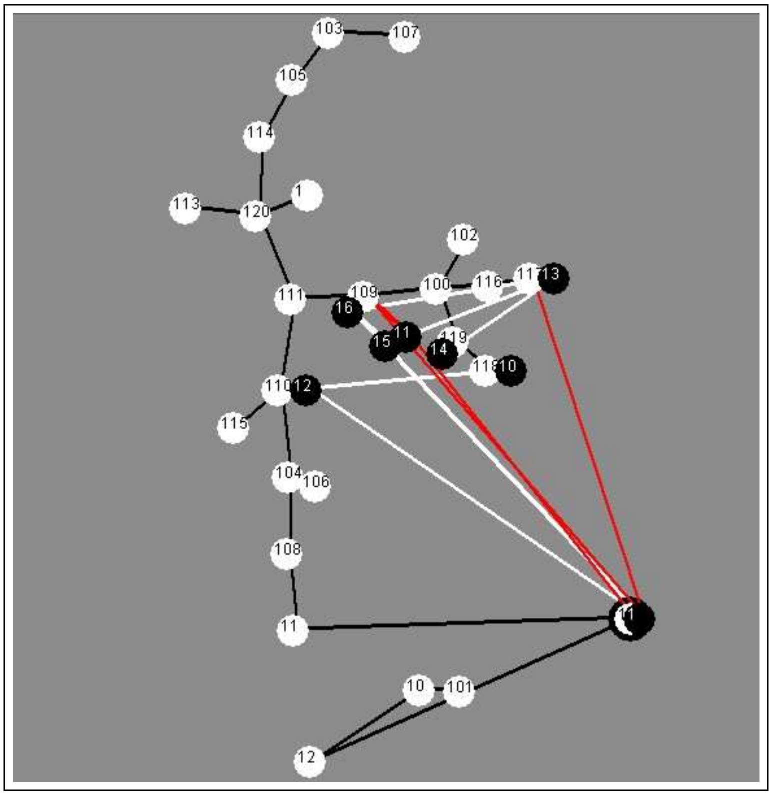


Figure 5.2: Joint layout influenced by index tree for threshold=0.5

with edges between them are still placed close to each other, then we can conclude that the group of similar matrices are affected by the same index node, or by index nodes that are most similar to each other. On the other hand, any exception to this behavior would be considered as a special observation that could provide new information relevant to the research on evolution.

Figure 5.2 shows a snapshot of the joint layout influenced by the index tree. The threshold is set to the default value of 0.5. We see that matrix node 1 is strongly influenced by index node 112 and that the former lies very close to the latter. Matrix node 1 has edges with both index nodes 109 and 117. However, since the computed inter-tree distance of matrix node 1 from index node 112 is the smallest (compared to the distance between matrix node

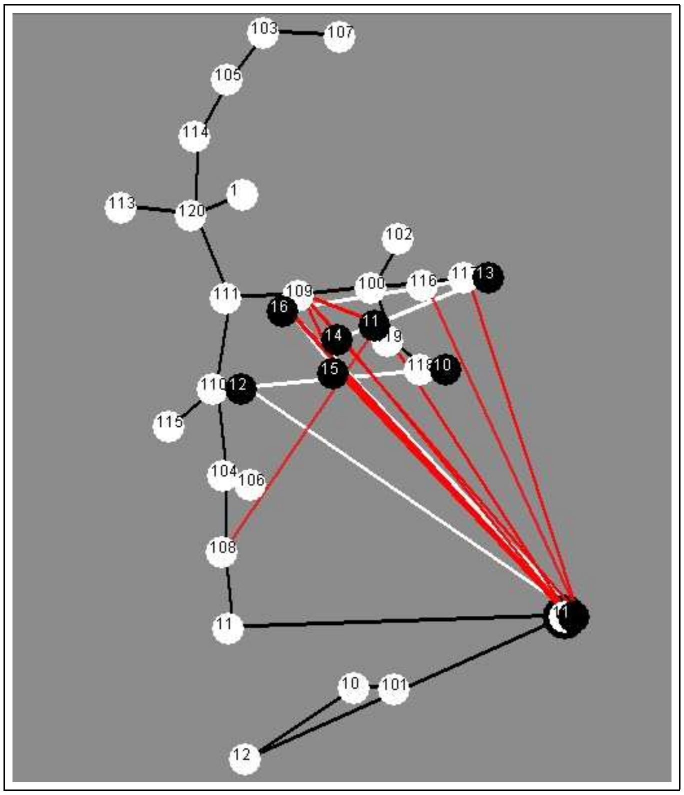


Figure 5.3: Joint layout influenced by index tree for threshold=0.58

1 and the other two index nodes), matrix 1 is placed close to index node 112. The same argument is true for matrix node 11, which is placed close to index node 109, although it shares an edge with both index nodes 109 and 112.

Figure 5.3 shows what happens if the threshold is slightly increased. We see that for a threshold of 0.58, several index-matrix edges are added to the graph. We also observe that matrix nodes 11, 14, 15, and 16 change their locations. This is because while placing the matrix nodes, we allow only those index nodes to influence the matrix nodes whose index-matrix distance is less than the threshold. Since the threshold has now changed, the location of the matrix node also changes. If we increase the threshold further to 0.75, we see more red edges and more changes in the locations of the matrix nodes. If two matrix

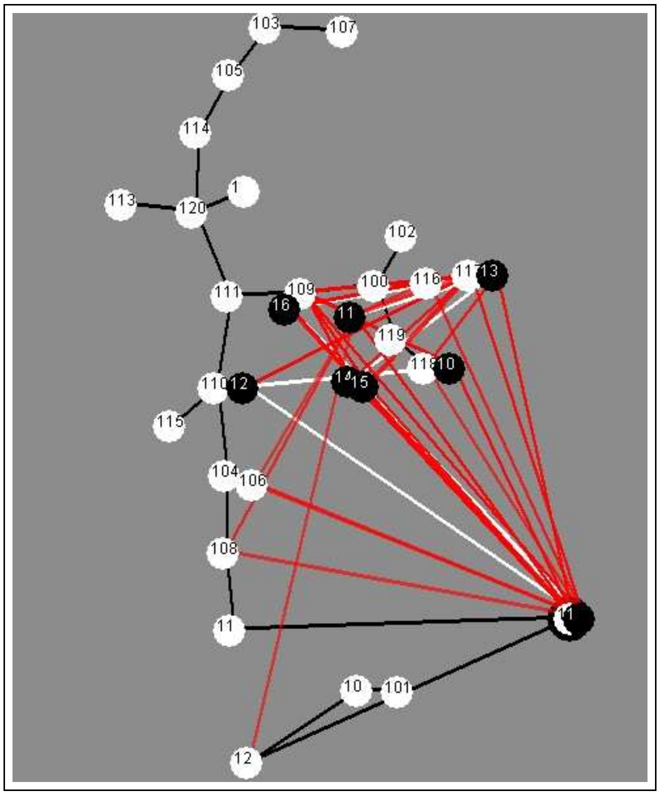


Figure 5.4: Joint layout influenced by index tree for threshold=0.75

nodes happen to be closest to the same index node and their individual distances to the index node are also same, then we place the matrix nodes radially around that index node in order to avoid overlapping.

The physical distance between the index node and each of the matrix nodes is a function of the computed inter-tree distance between them. Figure 5.5 shows the graph structure when the threshold is increased to the maximum value of 1.0. We see that all the index-matrix edges are visible now. Due to opacity mapping, however, only the significant ones are fully opaque, whereas the ones with higher computed inter-tree distances are partially transparent.

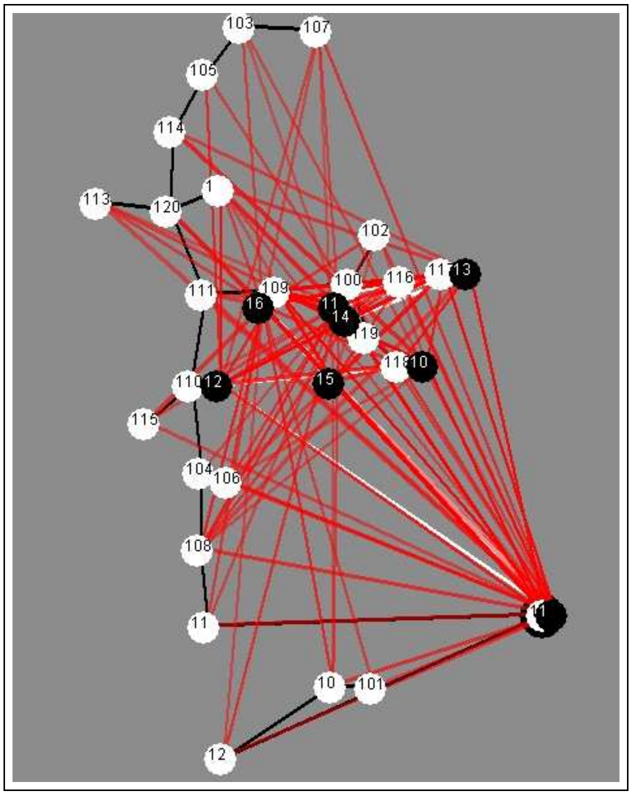


Figure 5.5: Joint layout influenced by index tree for threshold=1.0

5.1.3 Joint Layout Influenced by the Matrix Tree

The technique for this layout is similar to that of the previous section, with the index nodes replaced by the matrix nodes and vice versa. This layout tells the user about the group of indices that influence a mutation matrix. Figure 5.6 shows the layout of the graph influenced by the matrix tree. According to our algorithm, we first lay out the black matrix nodes using the tree layout algorithm and then apply spring embedding. The matrix tree structure is preserved in this case, unlike the layout in the previous section. In the next step, we determine the positions of the index nodes based on their closest distances to the matrix nodes. As shown in Figure 5.6, both indices 104 and 105 are closest to the matrix node 26, while index node 1 is influenced by matrix node 12.

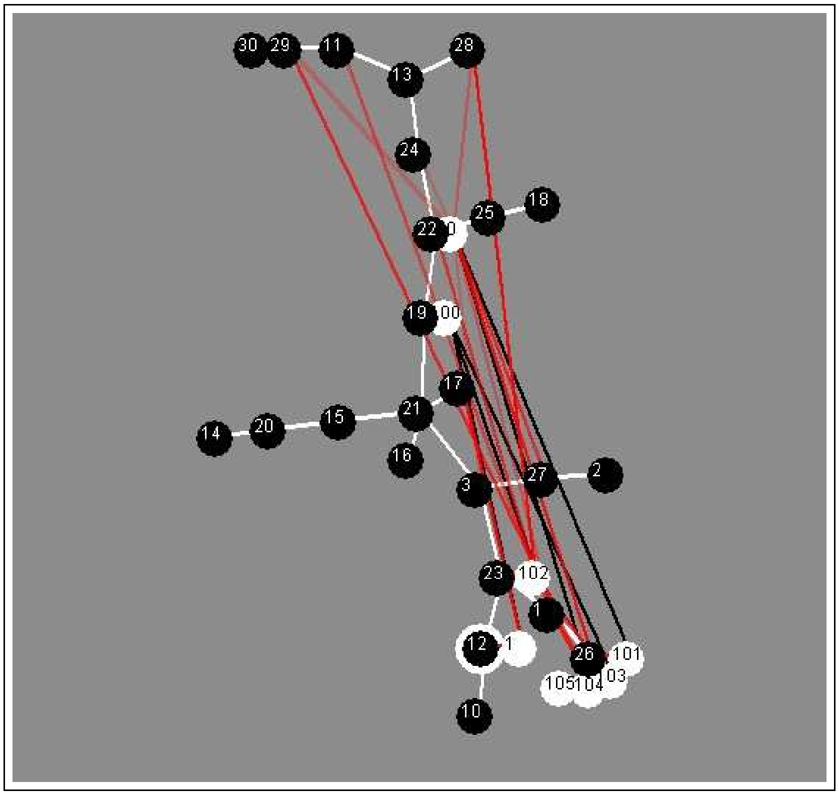


Figure 5.6: Joint layout influenced by index tree for threshold=0.5

In both the matrix-influenced and index-influenced joint layouts, zooming and graph movement are allowed as before. However, movement of individual nodes is not allowed in either case because that might destroy the meaningful information about the relative proximity of index and matrix nodes. For the index influenced layout, the user is only allowed to interactively move the index nodes, which simultaneously guides the movement of the matrix nodes, and vice versa for the matrix-influenced layout.

5.2 Evaluation of the Three Joint Layouts

The three joint layouts described in this chapter facilitate different tasks for the biologists. The effectiveness of the views can be measured by taking into consideration metrics such as the distance between an index node and its closest matrix node. Ideally, a matrix node that has the minimum computed inter-tree distance to an index node should be placed closest to it in screen space, and the physical distance should increase as the computed distance increases. This is where the matrix-based joint layout and the index-based joint layout outperform the joint independent layout. In the joint independent layout, the two spanning trees are first laid out on the screen independent of each other and spring embedding is applied to both independently, which gives them a perfect tree layout. Finally, an edge is drawn between each index-matrix pair if their distance is less than the threshold. This layout fails to draw the user's attention to the closest index matrix pairs and also results in many edge crossings. However, if the user is interested in the overall tree structures of the index spanning tree and the matrix spanning tree, then the joint independent layout would be a better view.

The claims of effectiveness of the above layouts are corroborated by the results of the experiments that we have conducted to evaluate the quality of our layout algorithm. For each of the layouts, our main goal is to find out if the difference in the edge lengths of any two matrix nodes, which are both adjacent to an index node, is greater than the human perceptible range. In order to experimentally verify this, we perform the following experiment. For the joint independent layout, we choose a set of index nodes and matrix nodes and

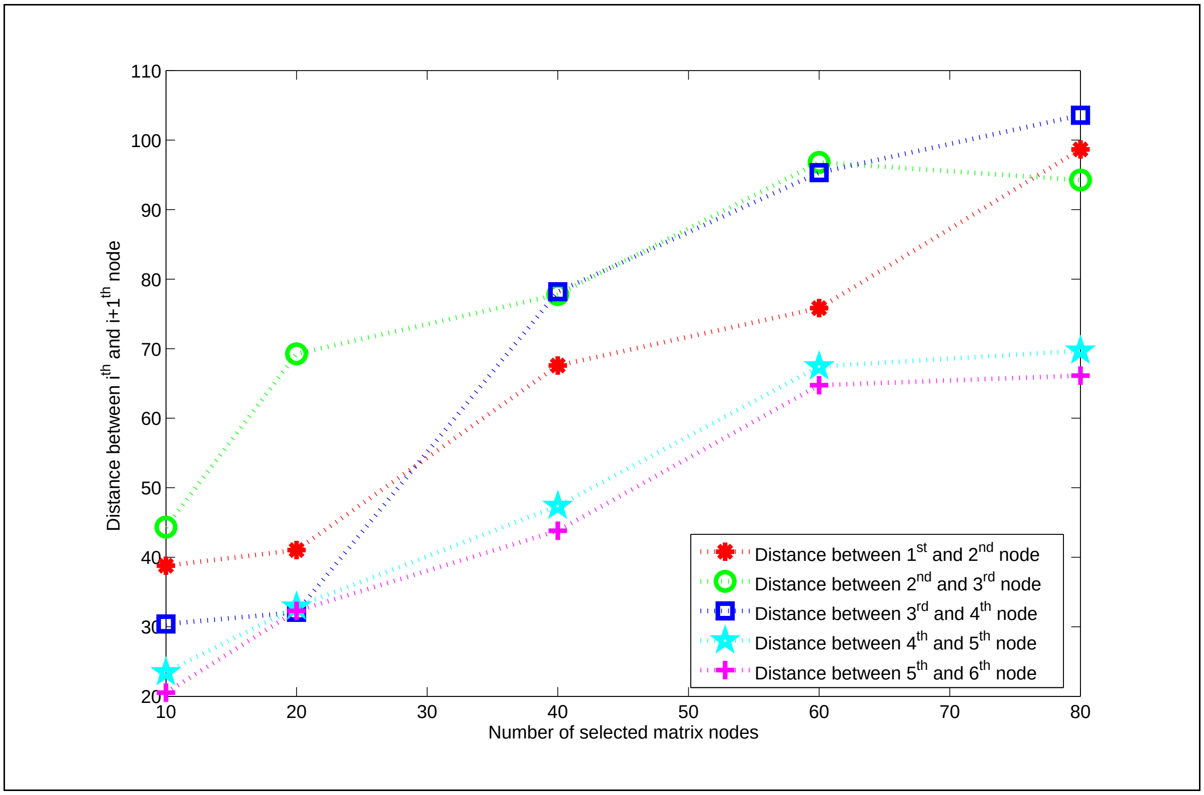


Figure 5.7: Experimental result showing the error between the first five indices (taken in sequence) with increasing number of matrices for the joint independent layout

lay them out. We select only those nodes for which the number of red edges (inter-tree edges) is five or more. We arrange these five red edges in ascending order based on their computed inter-tree distances and note the respective physical distances for each edge. We compute the difference between every consecutive pair of distances in this ordering. We take an average of these differences for every node in the tree with more than five edges, and plot these average differences against the number of selected nodes. We perform this experiment for different numbers of index and matrix nodes and report an average of ten runs for each choice of index and matrix nodes. It should be noted that the results reported here are for a threshold of 0.5.

Figure 5.7 shows the average pairwise difference in the edge lengths of five index-matrix edges, arranged in increasing order of the computed inter-tree distances. The horizontal axis shows an increasing number of nodes (both indices and matrices) for the joint independent layout. The individual lines denote the average difference in physical distance between the i^{th} and $(i+1)^{st}$ node. The nature of the graph seem to be random. This is expected in this layout, since the indices and matrices are placed randomly according to their own spanning tree structures, with no relationship with each other. With varying number of nodes, these lines have no specific pattern.

Figures 5.8 and 5.9 show the pairwise difference in the physical lengths between the lowest and second lowest weights, second lowest and third lowest and so on for five such pairs. The graph shows an increasing pattern as the total number of selected nodes is increased. The structure of the final layout is influenced by the index nodes in the first case and by the matrix nodes in the second case. We call these nodes, which determine the shape of the spanning trees, the *influencing* nodes, and the other type the *influenced* nodes. As we add more influencing nodes in the layout, the difference in physical distances increases because, in order to be placed closest to their corresponding influencing node, the influenced nodes are pulled apart from other influenced nodes. Similarly, more influencing nodes means that an influenced node can easily find a node closer to other influenced nodes, so the average difference in distance decreases.

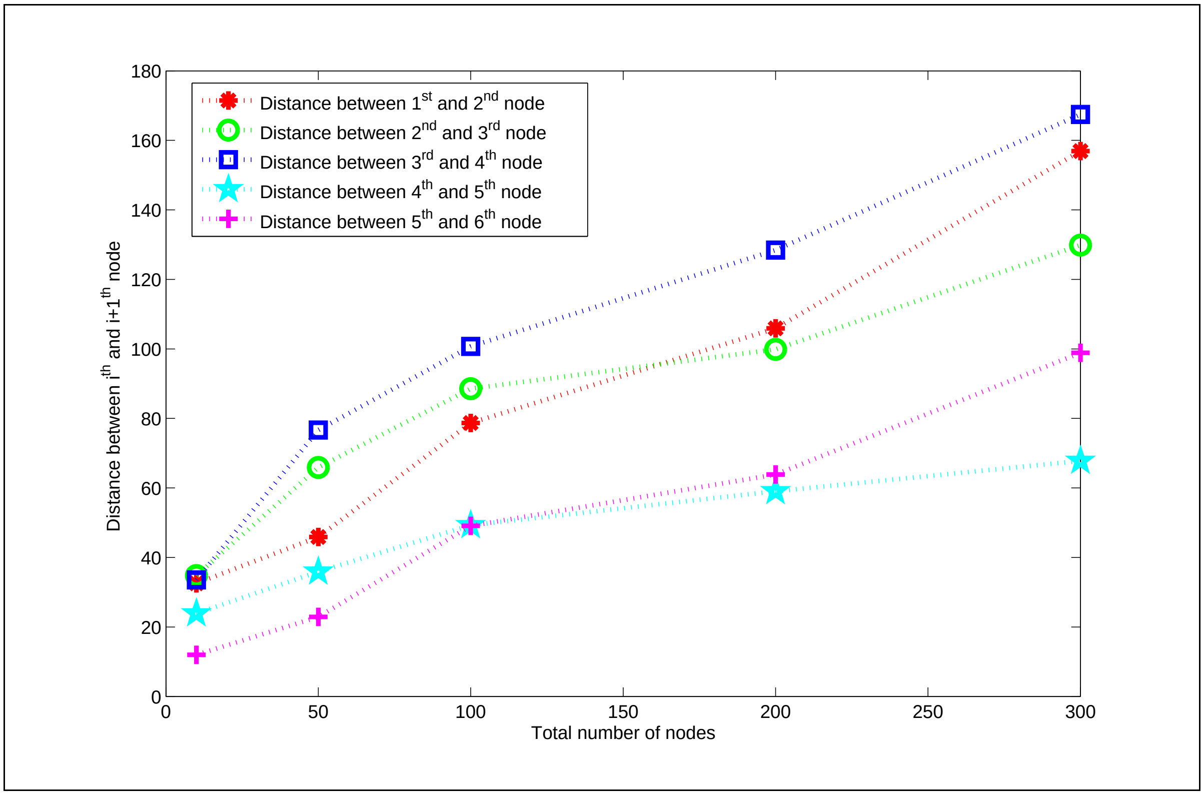


Figure 5.8: Experimental result showing the error between the first five indices (taken in sequence) with increasing number of matrices for the joint layout influenced by Index tree

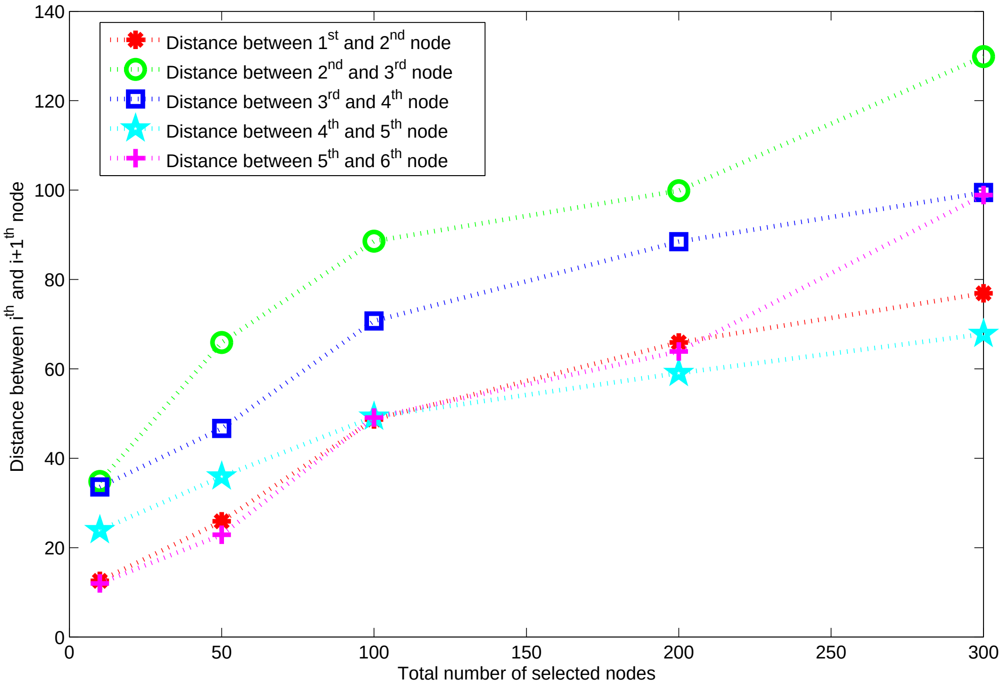


Figure 5.9: Experimental result showing the error between the first five indices (taken in sequence with increasing number of matrices for the joint layout influenced by Matrix tree

Chapter 6

Sparkler Arm Visualization Technique

In the previous chapter, we focused on the joint layout of the index and matrix spanning trees for their efficient comparison. In this chapter we present a different view of the data, which we call the *sparkler arm view*. This view, unlike the others described so far, is not graph visualization. This view helps the user to study the mutation matrices and compare them with respect to any one amino acid.

6.1 Multiple Query Visualization

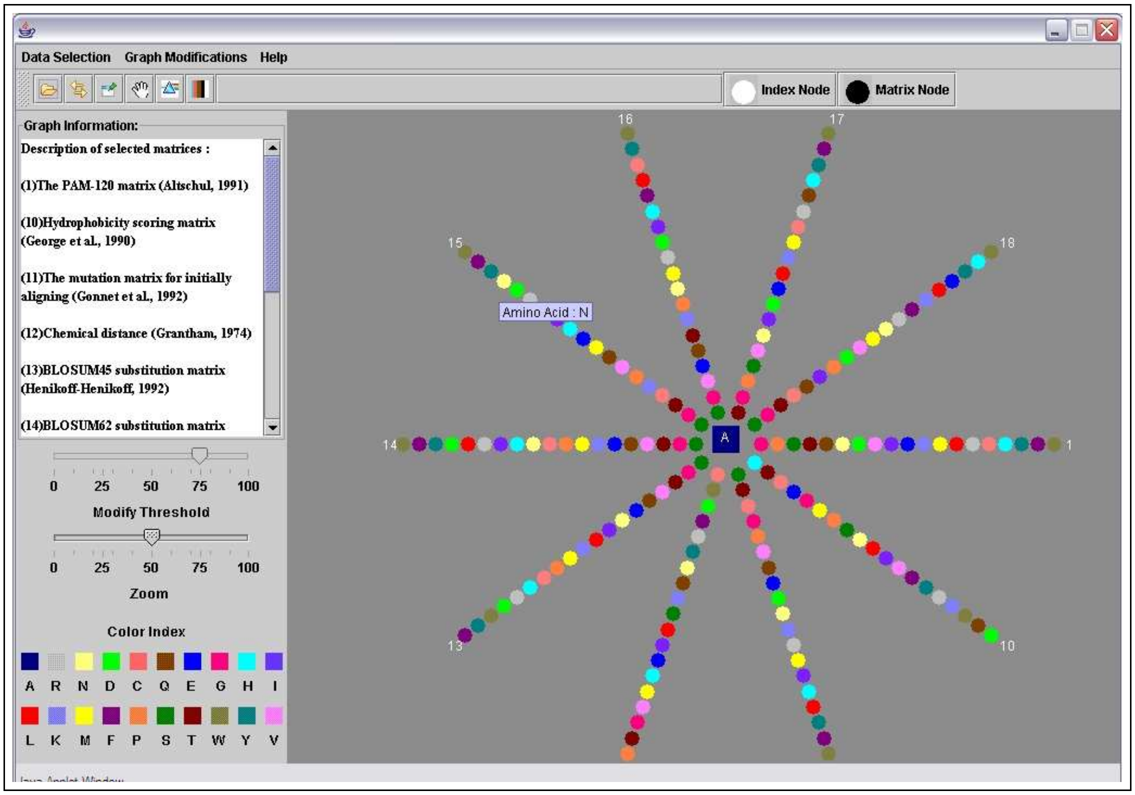
The sparkler arm visualization technique is inspired by the model proposed by Havre et al. [23] for displaying the results of multiple queries. Havre et al. describe a visualization that included a group of color-coded triangles in the center, where each such triangle represents a query from the user. The triangles are arranged in a circle covering an angle of 2π.

From each triangle, an *arm* extends radially outward and all results relevant to a triangle or query are arranged in order of importance along that arm. The query results that are most important or relevant are placed closer to the triangles at a smaller radius than the less important ones.

6.2 Sparkler Arm Technique

In this layout, the user first selects a particular amino acid and then a number of similarity matrices. The first step of the visualization algorithm is to take one of the chosen matrices and select the row corresponding to the chosen amino acid. The elements of that row are then arranged in descending order of magnitude. The values in the similarity matrices indicate how similar two amino acids are. The higher the similarity value, the higher the likelihood that one amino acid may substitute for the other during the course of evolution. After arranging the elements of the matrix row in descending order, we discard the first element, because the highest similarity of an amino acid is always with itself and it need not be considered for our visualization. We then choose the remaining nineteen elements and lay them out in a series of nineteen circles along a line, where each circle represents an amino acid. Thus for a selected set of n matrices, we have n such sparkler arms projecting from a central square representing the chosen amino acid.

In order to make the layout uniform, the sparkler arms are always placed equidistant from each other, so they are distributed evenly over the entire span of 2π degrees. Each amino acid is color-coded for our system. The color of the circle tells the user which amino acid it



Description of selected matrices :

(1)The PAM-120 matrix (Altschul, 1991)

(10)Hydrophobicity scoring matrix (George et al., 1990)

(11)The mutation matrix for initially aligning (Gonnet et al., 1992)

(12)Chemical distance (Grantham, 1974)

(13)BLOSUM45 substitution matrix (Henikoff-Henikoff, 1992)

(14)BLOSUM62 substitution matrix

Figure 6.1: Sparkler Arm View with A (Alanine) as the chosen Amino Acid and 10 Mutation Matrices

represents; the color index is always visible at the lower left panel to facilitate lookup. Also, if the user scrolls his mouse through the display, the names of the amino acids appear as tooltip texts. Each arm of the sparkler represents a matrix, with the matrix number printed at the end of the arm. The names of the selected matrices are simultaneously displayed in the graph information text panel. Figure 6.1 shows a sparkler arm view with ten selected matrices and amino acid **A** (Alanine) representing the central element.

If a particular amino acid is closest to the central amino acid in many of the arms of the sparkler arm view, then it can be inferred that, irrespective of the experimental conditions under which the similarity matrices have been computed, and irrespective of the metrics used for their computation, the amino acids are very similar. In Figure 6.1, we see that amino acid **S** is closest to **A** in six out of the ten sparkler arms. This means that **S** has a very high probability of substituting for the amino acid **A** during evolution. On the other hand, for a majority of the matrices chosen, if a particular amino acid is placed towards the end of the arm, then it indicates that the amino acid is dissimilar to the one at the center and hence it has a lower probability of replacing that amino acid during evolution. For example, in Figure 6.1, we observe that amino acid **W** is last in the sequence in seven of the ten arms of the sparkler.

6.3 Usefulness of the Sparkler Arm View

The sparkler arm view helps provide insight into the process of amino acid substitution. The similarity matrices represent different patterns of amino acid substitution. Some entries of these matrices are zero, implying that the corresponding pairs of amino acids never replace one another during evolution. A high value denotes a very high observed frequency of one amino acid replacing another. In a particular selection of matrices for the sparkler-arm view, if the same colors (representing amino acids) appear closer towards the central amino acid for a subgroup of matrices, then it can be inferred that the amino acid substitution pattern is similar for all those matrices. If the same colors appear towards the end of the arms for the other group of matrices, then these two sets of matrices might represent different types of organisms. Therefore, by observing the distribution of colors in different arms of the sparkler arm view, the user would be able to infer the similarity of mutation matrices. It would then be possible to account for such similarity by looking at the relationship among these matrices and the amino acid indices using the joint layout visualization scheme.

Chapter 7

Research Contribution and Future Work

The analysis and visualization of biological research data is currently an active topic of research in the bioinformatics community. This research is an addition to the existing literature of bioinformatics tools. However, this work has a considerable research contribution from the point of view of both biology and visualization.

7.1 Research Contribution in Biology

Research on genetic evolution has always concentrated on widening the knowledge-base of all possible patterns of amino acid substitution.In this research, we have developed a tool that might be able to help biologists understand and explain the observations of genetic evolution. Using this tool, amino acid substitution patterns are explained by associating the properties of amino acids with the varied substitution patterns found in nature. Once the biologists understand the mechanism of amino acid substitution, they might be able to predict new sequences containing amino acid pairs that have never substituted for one another before. This will help in rational design of proteins which is "*an attempt to modify protein molecules for specific applications by predicting which amino acid sequence will produce a protein with the desired properties*."

7.2 Research Contribution in Visualization

Although graph-based visualizations are widely used in many information visualization applications, we are visualizing a special kind of graph called a minimum spanning tree. Since a spanning tree does not have a well defined hierarchical structure, we choose a particular node as the root of the tree and lay out the rest of the tree based on that assumption using a new tree layout algorithm.

More significantly, in this research we compare two spanning trees having an unequal number of nodes. There is a considerable amount of literature in graph theory and graph visualization for comparing graphs. However, they all work under the assumption that the two graphs represent identical entities and have a comparable number of nodes. We do not base our research on any such assumption. We have compared two entirely different entities, namely, the amino acid indices and the mutation matrices. The number of indices or matrices in a single visualization may also differ. We have developed a novel graph visualization technique for accomplishing this task.

This thesis also provides an easy and simple way of representing 20-dimensional data in two dimensions by using the banded edges view in the index spanning tree. Finally, the sparkler-arm view, although not a new visualization scheme, is used as a distance measuring tool, unlike the original work that describes the sparkler arm.

Our visualizations can provide new insights into research on genetic evolution. Currently, genetic evolution is explained in terms of already observed evolutionary patterns. However, our tool would enable researchers to explore genetic evolution as a cause-and-effect model. The physical and chemical properties of these amino acids would act as the cause of amino acid substitutions generating new protein molecules, which is the fundamental theory behind genetic evolution. This visualization algorithm, on the other hand, can be extended to visualize any database of related or unrelated elements. It can also be used to visualize results of cluster analysis for any data mining algorithm.

7.3 Future Work

This research establishes many new ideas and it would be interesting to study the strengths and weaknesses of these methods. For example, the amino acid indices are linearly normalized values of the original properties, which can have either linear or non-linear distributions. It would be interesting to study the effect of the linear normalization on the results produced by this tool, especially when the original distribution of the amino acid property is non-linear. Again, the matrices representing the amino acid substitution patterns are generated after applying different mathematical operations on the probability matrices.

These mathematical operations range from simple normalization to log-odds calculations. It would be useful to know the physical interpretation of the distance metric between the amino acid indices and the mutation matrices. A user study on the usefulness of the different visualization schemes would enable us to understand the usefulness of this tool in performing specific tasks. Finally, it would be interesting to know how well correlation coefficient performs as a distance metric and what other alternatives exist in this scenario.

Several improvements to the tool could be made. The visualization scheme currently used is 2D. However, it could be extended to 3D if the number of independent visual parameters increases. Also, different visual cues such as hue mapping, brightness, and saturation mapping could be used on both the nodes and the edges if we want to capture more dimensions of the data in the 2D screen space.

The existing technique computes a minimum spanning tree for depicting the closest elements of a graph. However, this spanning tree view is only able to tell the user about the closest pairs of nodes in the graph. If the user wants to know the distance between a pair of nodes in the spanning tree that do not have an edge between them, the current version of the spanning tree visualization fails to give that information. To overcome this problem, we can cluster the nodes of the spanning tree into a number of different groups based on their properties. Color coding the different clusters would enable us to draw inferences about the relationships between unconnected nodes in the spanning tree.

7.4 Conclusion

Interpreting relationships among different entities belonging to different databases is a challenging task. This tool enables biologists to construct interactively a relationship between a chosen group of indices and matrices. This not only allows them to visualize 20-dimensional data in 2D with ease and clarity, but also allows them to draw conclusions about the ways in which the amino acid indices might have shaped the similarity matrices. We use graph visualization combined with a geometric primitive-based sparkler arm technique for representing these relationships effectively.

Bibliography

[1] Japan AAIndex Database. http://www.genome.ad.jp/dbget/aaindex.html.

[2] Nina Amenta and Jeff Klingner. Case study: Visualizing sets of evolutionary trees. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)*, pages 71–73, 2002.

[3] Jesse Bentz, Albion Baucom, Marc Hansen, and Lydia Gregoret. Dinamo: An interactive protein alignment and model building tool. *Bioinformatics*, 15:309–316, 1999.

[4] BIOINFORMATICS. http://en.wikipedia.org/wiki/bioinformatics.

[5] BLAST. http://www.ncbi.nih.gov/blast/.

[6] S. K. Card, J. D. Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1999.

[7] Mathew Clement, Margaret Ellis, Josh Steele, Yuying Tian, and Chris North. Gene expression mural: Visualizing gene expression databases. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'01)*, pages 71–73, 2001.

[8] Mathworld Correlation Coeff. mathworld.wolfram.com/correlationcoefficient.html.

[9] Protein Sequence Database. http://pir.georgetown.edu/pirwww/search/textpsd.shtml.

[10] UMBC AAIndex Database. http://phenotype.biosci.umbc.edu:8080/aaindex/.

[11] N. Deo. *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall, 2000.

[12] E.C. deVerdière, M. Pocchiola, and G.Vegter. Tutte's barycenter method applied to isotopies. *Journal of Computational Geometry*, 26:81–97, 2003.

[13] G. diBattista, P. Eades, R. Tamasia, and I. Tollis. *Graph Drawing Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[14] G. diBattista, G. Liotta, and F. Vargiu. Spirality and optimal orthogonal drawings. *SIAM Journal on Computing*, 27:1764–1811, 1998.

[15] P. Eades. Drawing free trees. In *Bulletin of the Institute for Combinatorics and its Applications*, pages 10–36, 1992.

[16] Stephen J. Freeland. The Darwinian genetic code: An adaptation for adapting? *Genetic Programming and Evolvable Machines*, 3:113–127, 2002.

[17] Michael Y. Galperin. The molecular biology database collection: 2004 update. *Nucleic Acids Research*, 32, 2004.

[18] R. Grantham. Amino acid difference formula to help explain protein evolution. *Science*, 6:862–864, 1974.

[19] Marc Hansen, Jesse Bentz, Albion Baucom, and Lydia Gregoret. Dinamo: A coupled sequence alignment editor/molecular graphics tool for interactive homology modeling of proteins. In *Pacific Symposium on Biocomputing*, pages 106–117, 1998.

[20] Marc Hansen, Doanna Meads, and Alex Pang. Comparative visualization of structure-sequence alignments. In *IEEE Symposium on Information Visualization*, October 1998.

[21] Marc D. Hansen, Erik Charp, Suresh Lodha, Doanna Meads, and Alex Pang. Promuse: A system for multi-media data presentation of protein structural alignments. In *Pacific Symposium on Biocomputing*, pages 368–379, 1999.

[22] Marc D. Hansen, Suresh Lodha, and Alex Pang. Profeel: Low cost visual-haptic perceptualization of protein structure-structure alignments. In *Pacific Symposium on Biocomputing*, pages 218–229, 2000.

[23] Susan Havre, Elizabeth Hetzler, Ken Perrine, Elizabeth Jurrus, and Nancy Miller. Interactive visualization of multiple query results. In *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, pages 105–111, 2001.

[24] Ivan Herman, Guy Melancon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. In *IEEE Transactions on Visualization and Computer Graphics*, volume 6, pages 24–43, January 2000.

[25] Ata Kabn, Peter Tino, and Mark Girolami. General framework for a principled hierarchical visualization of high-dimensional data. In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July 2002.

[26] S. Kawashima and M. Kanehisa. Aaindex: Amino acid index database. *Nucleic Acids Research*, 28, 2000.

[27] Kiran Lakkaraju, Ratna Bearavolu, and William Yurcik. Nvisionip - a traffic visualization tool for security analysis of large and complex networks. In *International Multiconference on Measurement, Modelling, and Evaluation of Computer-Communication Systems*, September 2003.

[28] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, 1986.

[29] James Xinzhi Li. Visualization of high-dimensional data with relational perspective map. *Information Visualization*, 3:49–59, 2004.

[30] Doanna Meads, Marc D. Hansen, and Alex Pang. Protalign: A 3-dimensional protein alignment assessment tool. In *Pacific Symposium on Biocomputing*, pages 354–367, 1999.

[31] G. Melanon and I. Herman. Circular drawings of rooted trees. In *Reports of the Centre for Mathematics and Computer Sciences*, 1998.

[32] Tamara Munzner, Francois Guimbretiere, Serdar Tasiran, Li Zhang, and Yunhong Zhou. Treejuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility. In *Proceedings of ACM SIGGRAPH 2003*, pages 453–462, 2003.

[33] Gosta Nachman. http://www.dina.dk/dinanews/dinanews3.htm.

[34] Chris North and Ben Shneiderman. Snap-together visualization: A user interface for coordinating visualizations via relational schemata. In *Proc. Advanced Visual Interfaces*, May 2000.

[35] Resolution of Vision. http://ocw.mit.edu/nr/rdonlyres/.

[36] S. Pook, G. Vaysseix, and E. Barillot. Zomit: Biological data visualization and browsing. *Bioinformatics*, 14:807–814, July 1998.

[37] E.M. Reingold and J.S. Tilford. Tidier drawing of trees. *IEEE Transactions on Software Engineering*, 7:223–228, 1981.

[38] Human Genome Resources. http://www.ncbi.nlm.nih.gov/genome/guide/human/.

[39] U. Rost and E. Bornberg-Bauer. Treewiz: Interactive exploration of huge trees. *Bioinformatics*, 18:109–114, 2002.

[40] E. Segal, A. Kaushal, R. Yelensky, T. Pham1, A. Regev, D. Koller, and N. Friedman. Genexpress: A visualization and statistical analysis tool for gene expression and sequence data. In *Proceedings of the 11th Inter. Conf. on Intelligent Systems for Molecular Biology (ISMB)*, 2004.

[41] Y. Shiloach. PhD Thesis: Arrangements of planar graphs on the planar lattices. 1976.

[42] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11:109–125, 1981.

[43] K Tomii and M Kanehisa. Analysis of amino acid indices and mutation matrices for sequence comparison and structure prediction of proteins. *Protein Engineering*, 9:27–36, January 1996.

[44] LLC TOUCHGRAPH. http://www.touchgraph.com.

[45] Greg Turk and David Banks. Image-guided streamline placement. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 453–460. ACM Press, 1996.

[46] Jorg A. Walter and Helge Ritter. On interactive visualization of highdimensional data using the hyperbolic plane. In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July 2002.

[47] Dr. Stephen Freeland's web page. http://phenotype.biosci.umbc.edu.

[48] L. Y. Yampolsky and A. Stoltzfus. The exchangeability of amino acids in proteins. *Genetics*, 4:1459–1472, 2005.