

**PROTERAN: ANIMATED TERRAIN EVOLUTION FOR
VISUAL ANALYSIS OF PROTEIN FOLDING
TRAJECTORY**

Kush Kapila

A Thesis

in

The Department of

Computer Science

Presented in partial fulfillment of the requirements for

the degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

August 2004

© Kush Kapila, 2004



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*

ISBN: 0-612-94743-2

Our file *Notre référence*

ISBN: 0-612-94743-2

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

ABSTRACT

PROTERAN: Animated Terrain Evolution for Visual Analysis of Protein Folding Trajectory

Kush Kapila

In the field of bio-informatics, the analysis of voluminous data is becoming increasingly crucial to understanding the underlying biology and answering important questions. Various clustering techniques such as Hierarchical, SOMs, K-means and PCA are being used to cluster gene expression data to find the functions of unknown genes. Even these sophisticated algorithms are futile if the results are not appropriately interpreted, thus visualization techniques play an important role in analyzing data. Similar to clustering of gene expression data is that of clustering the characteristics of protein folding trajectory and a good visualization tool can help visual analysis and can provide faster and deeper insights into the manner in which a protein folds.

With protein characteristics data and specific visualization requirements provided by Dr. Laxmi Parida and Dr. Ruhong Zhou of the Computation Biology Group at the IBM T. J. Watson Research Center, a new 3D visualization technique was designed and developed. This customized technique helps identify the major states a protein folds into through the use of an animated terrain. This technique was implemented as part of the interactive visualization program PROTERAN and tested with the β -Hairpin clustered data provided.

Acknowledgements

I would like to express my sincerest appreciation to all those who helped in the completion of this thesis. I would especially like to thank my supervisor, Dr. S.P. Mudur for his knowledge, his encouragement and his guidance during my research. For this I am forever indebted to him.

I would also like to thank Dr. Laxmi Parida and Dr. Ruhong Zhou of the Computational Biology Group at the IBM T. J. Watson Research Center for their patience and giving me the opportunity to work with them.

I would finally like to thank my family (Mom, Dad, Aarti, Kamal, Jij, Bhabhi, Pooja and Neel) for their support. Last but not least I would like to thank my fiancée, Sandhya, without her this thesis would not have been possible.

TABLE OF CONTENTS

CHAPTER 1 – Introduction.....	1
1.1 Genes & Proteins	1
1.2 Microarrays (Gene-Chips/Protein Chips).....	4
1.2.1 Gene-Chips	4
1.2.2 Protein Chips.....	7
1.3 Clustering of data.....	8
1.4 Visualization of Clusters.....	8
1.5 Research Problem	9
1.6 Objectives of this Research.....	10
1.7 Overview of Thesis	10
CHAPTER 2 - Clustering and Visualization Techniques.....	12
2.1 Expression Matrix.....	12
2.1.1 Data Adjustments.....	13
2.2 Similarity Metrics	13
2.3 Clustering Methods.....	14
2.4 Hierarchical Clustering.....	14
2.4.1 Algorithm.....	15
2.4.2 Visualization	17
2.4.3 Advantages.....	18
2.4.4 Disadvantages	19
2.5 Self Organizing Maps (SOMs)	20
2.5.1 Algorithm.....	20
2.5.2 Visualization	22
2.5.3 Advantages.....	25
2.5.4 Disadvantages	26

2.6	K-Means.....	26
2.6.1	Algorithm.....	27
2.6.2	Visualization	30
2.6.3	Advantages.....	30
2.6.4	Disadvantages	31
2.7	Principal Component Analysis (PCA).....	31
2.7.1	Algorithm.....	32
2.7.2	Visualization	34
2.7.3	Advantages.....	34
2.7.4	Disadvantages	35
2.8	Biclustering.....	35
2.8.1	Algorithm.....	36
2.8.2	Observations	38
2.9	Available Software.....	38
CHAPTER 3 - Protein Folding Trajectory Analysis		40
3.1	Data.....	41
3.2	Patterned Clusters	44
3.2.1	The Need for Visually Analyzing the Patterned Cluster Data.....	46
3.2.2	Program Requirements.....	47
3.2.2.1	Global View:.....	47
3.2.2.2	Navigation and focus:	47
3.2.2.3	Relative growth rates:	48
3.2.2.4	Detailed Query:.....	48
3.3	The need for a Custom Visualization Tool.....	48
CHAPTER 4 - PROTERAN.....		50
4.1	Terrain Metaphor	50

4.2	Mapping of Patterned Cluster Data into Terrain Geometry.....	51
4.2.1	Patterned Cluster Layout.....	54
4.2.2	Animated Terrain Evolution	58
4.2.3	Interaction Facilities.....	58
4.3	Programming Tools	59
4.4	PROTERAN	60
4.4.1	Opening Screen.....	60
4.4.1.1	‘Filter File’ Button	60
4.4.1.2	‘Processed File’ Button.....	60
4.4.1.3	‘Quit’ button.....	60
4.4.2	Main Screen	61
4.4.2.1	Fly Through and Landing on Mountain:.....	62
4.4.2.2	Data Caption:	63
4.4.2.2.1	Pattern:	63
4.4.2.2.2	Hill Number:	63
4.4.2.2.3	Column x:.....	63
4.4.2.2.4	Height:.....	63
4.4.2.2.5	Time:.....	64
4.4.2.2.6	FPS:.....	64
4.4.2.3	Buttons:.....	64
4.4.2.3.1	Animation	64
4.4.2.3.1.1	Start:.....	64
4.4.2.3.1.2	Stop:.....	65
4.4.2.3.2	Panning:	65
4.4.2.3.2.1	Left/Right:.....	65
4.4.2.3.2.2	Forward/Back:.....	65
4.4.2.3.2.3	Stop Panning:.....	65
4.4.2.3.3	Quick View:.....	65
4.4.2.3.3.1	Top & Bottom Views:.....	65

4.4.2.3.3.2	Pattern 2-7:.....	65
4.4.2.3.4	Miscellaneous:	65
4.4.2.3.4.1	Save Image:.....	65
4.4.2.3.4.2	Quit:	66
4.4.3	Other features:.....	66
4.5	Testing and Release	67
CHAPTER 5	- Conclusions and Extensions.....	68
5.1	Initial Experience with PROTERAN.....	68
5.2	Extensions.....	69
References	71
Appendix	75
APPENDIX A	75
APPENDIX B	77

LIST OF FIGURES

Figure 1-1:	DNA double helix structure	2
Figure 1-2:	Nucleotide structure.....	2
Figure 1-3:	Microarray.....	5
Figure 1-4:	High-speed robot used to imprint cDNA strands onto a slide	6
Figure 1-5:	Process of creating a microarray.....	7
Figure 2-1:	Steps 1-4 of Agglomerative Hierarchical Clustering.....	16
Figure 2-2:	Dendrogram 1	17
Figure 2-3:	Dendrogram 2	17
Figure 2-4:	Dendrogram 3	17
Figure 2-5:	Dendrogram and reordered expression matrix.....	19
Figure 2-6:	Principle of SOMs.....	22
Figure 2-7:	Hierarchical Clustering of SOM subsets.....	23
Figure 2-8:	Error bar representation of gene expression data.....	24
Figure 2-9:	U-matrix representation of the SOM	24
Figure 2-10:	Neurons in a SOM.	25
Figure 2-11:	Example of K-means, $k = 2$	29
Figure 2-12:	The results of the k-means algorithm with different k-values.	31
Figure 2-13:	Principal component analysis	32
Figure 2-14:	Visualization of PCA on gene expression.	34
Figure 2-15:	Biclustering versus traditional clustering.....	37
Figure 3-1:	A hypothetical state diagram of a folding protein.	41
Figure 3-2:	Patterned cluster.....	44
Figure 4-1:	PROTERAN	51
Figure 4-2:	Layout of Pattern Types on the Ground Plane.....	55
Figure 4-3:	Layout of Column Combinations.....	56
Figure 4-4:	Opening screen of PROTERAN	61
Figure 4-5:	Main Screen of PROTERAN.....	62
Figure 4-6:	Data Caption	63
Figure 4-7:	Buttons used in PROTERAN.....	64

LIST OF TABLES

Table 1-1:	The 20 amino acids and their codons.....	3
Table 2-1:	Expression Matrix.....	12
Table 2-2:	Some Image analysis and Data Mining Software Packages	39
Table 3-1:	Sample data of β -hairpin's reaction coordinates over time.	43
Table 3-2:	Threshold added to each characteristic.	43
Table 3-3:	Results after adding threshold.....	44
Table 3-4:	Sample patterned cluster file.....	46
Table 4-1:	Patterns of type 2	53
Table 4-2:	Number of permutations of each pattern type.....	54
Table 4-3:	Pattern 2 with combination 01	57
Table 4-4:	List of the Pattern Type Colors.....	66

CHAPTER 1 – Introduction

With the recent fusion of biology and computer science known as the discipline of bioinformatics, the last few decades have shown an explosion in the availability of biological data for the scientific community. This can be seen with the recent completion of the Human Genome Project where almost 95% of the genome, consisting of 6 billion nucleotides, has been mapped giving biologists a master blueprint for all the cellular structures. The analysis of this data is crucial to understanding the underlying biology and for answering vital questions. Various techniques such as data mining and clustering are being used on large data to extract useful information. Even algorithms and analysis techniques would not be useful if the results are difficult to interpret or are not appropriately interpreted: thus visualization techniques play an important role in bridging this gap. Through the use of visualization techniques, scientists are able to understand their results better as well as get a different perspective of the data that would otherwise be lost in a very large text file.

1.1 Genes & Proteins

The complete set of instructions for making an organism is called its genome and is found in every nucleus of a person's trillions of cells [1]. This blueprint is organized in the form of 23 chromosomes, which are made up of entwined deoxyribonucleic acid (DNA) [1, 2]. DNA is a double helix structure (*cf.* Figures 1-1) of repeating nucleotide base pairs, with each nucleotide containing one phosphate group, one sugar and one base (guanine, thymine, cytosine or adenine) (*cf.* Figure 1-2). A sequence of three nucleotides

constitutes a codon, which is the protein coding vocabulary. The sequence of codons in a gene specifies the amino-acid sequence of the protein it encodes (*cf.* Table 1-1) [3].

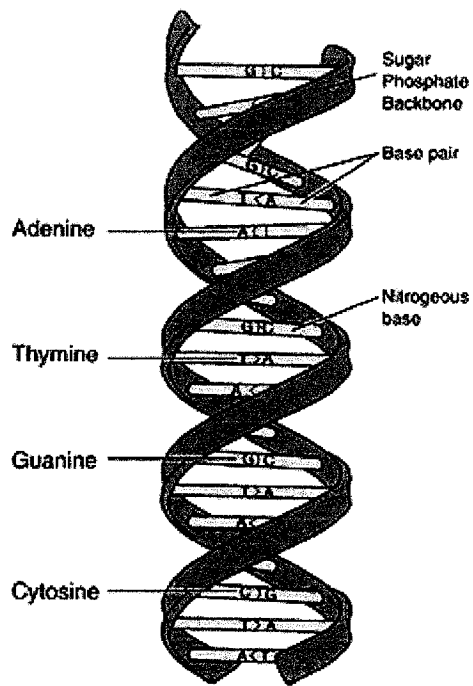


Figure 1-1: DNA double helix structure

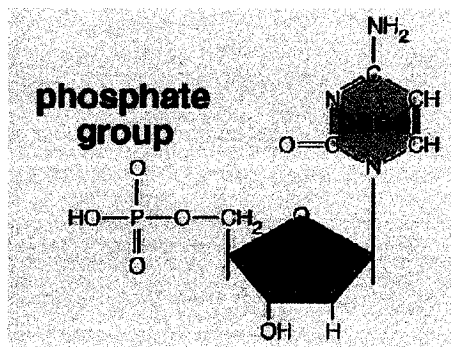


Figure 1-2: Nucleotide structure consists of one phosphate group, one sugar and one base (guanine, thymine, cytosine or adenine).

Table 1-1: The 20 amino acids used in proteins and the codons that code for each of those amino acids.

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

- Ala, Arg, Asn etc. are shortened names of Amino acids. The full names can be found in any text book on biology [2].

- A, G, C, and U are shorthand for nucleotides Adenine, Guanine, Cytosine and Uracil found in Ribonucleic Acid (RNA). RNA is a form of genetic information, involved in the translation process from DNA into proteins. A combination of any of these three nucleotides makes a codon which specifies an amino acid [2].

Through the proteins they encode, genes govern the cells in which they reside. In multicellular organisms they control development of the individual from the fertilized egg and the day-to-day functions of the cells that make up tissues and organs. The instrumental roles of their protein products range from mechanical support of the cell structure to the transportation and manufacture of other molecules and to the regulation of other proteins' activities [3].

Although there are roughly 3 billion nucleotide base pairs only 10% of the genome is known to include the protein-coding sequences called genes [1]. Furthermore even though most of the Human Genome has been mapped and the location of these genes has been found what still remains to be discovered are the functions of the proteins they code for. Knowing the function of these genes and their proteins could lead to improved drugs and cures to many diseases.

1.2 Microarrays (Gene-Chips/Protein Chips)

1.2.1 Gene-Chips

It is these thousands of genes and their corresponding proteins that work together in a complex fashion to perform the everyday functions of an organism. Traditionally, experiments on genes and proteins have been performed on a one-by-one basis. Thus the throughput was limited and a global view hard to achieve. This problem was solved with the advent of microarrays (*cf.* Figure 1-3) or better known by Affymetrix's trademark name, Gene Chips. Usually made on glass but sometimes on nylon substrates, microarrays enable scientists to analyze genes on a genomic level [4].

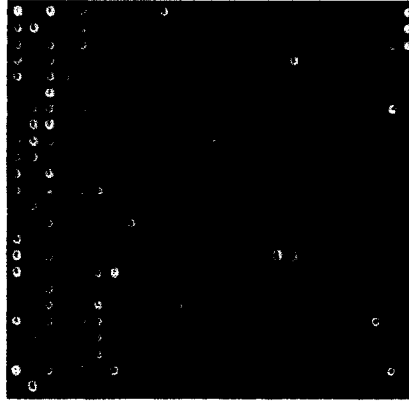


Figure 1-3: Microarray¹

Using high-speed robots, like the one depicted in Figure 1-4, thousands of complimentary DNA (cDNA) strands can be printed onto a slide. cDNA is similar to DNA except it is single stranded. Each of these cDNA strands called spots represent a gene; an entire genome can be printed on the standard microarray size of 1x3 inches [5]. Cells are then allowed to grow in regular conditions called the control, and experimental conditions called the sample. The mRNA from each of these types of cells is extracted (*cf.* Figure 1-5 (A)).

Each mRNA is then converted to fluorescently labeled cDNA by incorporating either Cy3 or Cy5-dUTP using a single round of reverse transcription (*cf.* Figure 1-5 (B, C)) [6, 7]. The control sample is labeled with Cy3 dye and the sample with Cy5 that are green and red colors respectively. Both types of cDNA are mixed together and then placed on the microarray to incubate with the spots on the chip (*cf.* Figure 1-5 (D)). Each cDNA has a corresponding spot on the chip to attach itself to. Four possibilities could occur with a spot (gene) when the microarray is allowed to incubate with the cDNA.

1. Only control cDNA attaches itself to the gene

¹ <http://www.dpz.gwdg.de/clineu/people/abumaria/bild.jpg>

2. Only sample cDNA attaches itself to the gene
3. Both types of cDNA (Red, Green) attach themselves to the gene
4. Neither the control nor the sample cDNA attach to the gene

Any unbound cDNA is then washed off and the microarray is placed in a black box. The microarray is then passed over with a green laser that creates an image of only the spots with green cDNA attached to it. This image is stored for later use. The same thing occurs with a red laser, this time only spots with red cDNA are stored.

By means of software, both images are then superimposed on each other. Spots with more control cDNA will appear green while spots that contain more sample cDNA will appear red. Spots with an equal amount of both types of cDNA will appear yellow (green mixed with red). Finally spots without any cDNA will appear black. By examining the fluorescence intensities we can compare the relative expression levels of thousands of genes in a control versus sample condition.



Figure 1-4: High-speed robot used to imprint cDNA strands onto a slide²

² http://www.research.vt.edu/resmag/resmag2001/s_marroof_lab.html

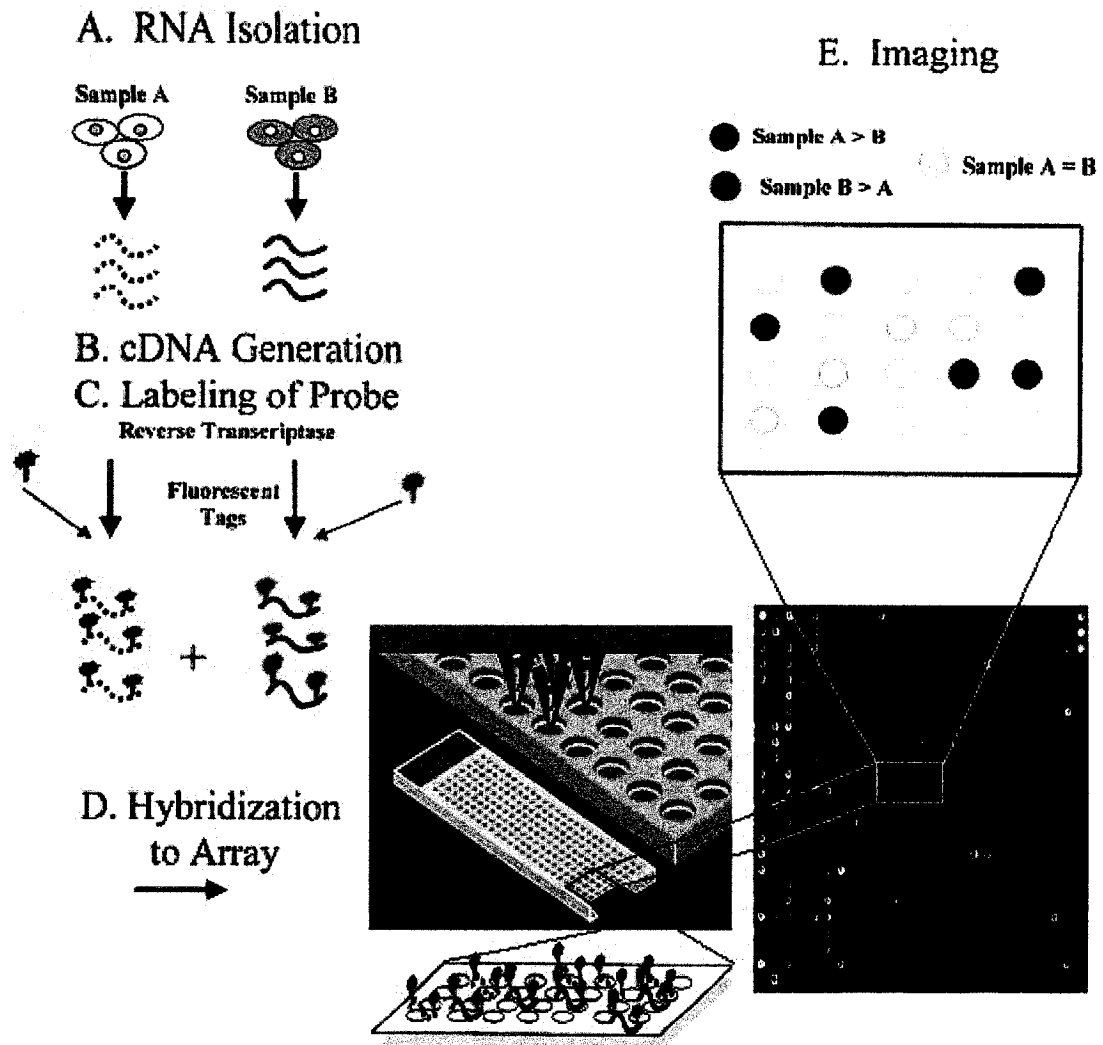


Figure 1-5: Process of creating a microarray³

1.2.2 Protein Chips

In addition to studying genes through microarray experiments, proteins can also be analyzed using protein chips [4]. With more than 3 million proteins versus some 40,000 genes there may be more of a need for protein chips.

Protein chips fall into two categories:

³ <http://www.fao.org/DOCREP/003/X6884E/x6884e00.jpg>

1. **Capture chips** - Count the number of proteins in a sample.
2. **Interaction chips** - Look at protein-protein interactions.

Protein chips are a relatively new technology and their 3D nature along with their tendency to denature on contact with the chip makes their creation far more difficult than microarrays [8].

Gene-chips and protein chips are however just the first step to finding the functions of unknown genes and the proteins they produce.

1.3 Clustering of data

A key step in the analysis of gene expression data is the identification of groups of genes that manifest similar expression patterns [9]. Complex algorithms are used to group together genes of unknown functions with those whose functions are known. This is done in an attempt to learn more about the gene whose functions are unknown. This process of organizing genes into biologically relevant groups is called clustering and there are three reasons for this [7].

1. Genes that share similar expression patterns are said to be coexpressed. Evidence supports that many functionally related genes are coexpressed.
2. Coexpressed genes can reveal much about regulatory systems
3. Gene expression varies greatly in different cell types and states

1.4 Visualization of Clusters

While clustering algorithms have proven to be useful as a first pass in finding more information about genes and their proteins, without a proper visualization of the

clustered data, valuable information is lost. With thousands of clusters, scientists lose the overall picture of the data. In addition, the relationship between individual clusters is hard to determine, if presented to the scientist in textual formats.

A good visual representation should allow scientists to keep a global view as well as observe the relationship between clusters. A visual representation can also give biologists a different perspective on the data.

1.5 Research Problem

Understanding how a protein folds into a functional or structural configuration is arguably one of the most important and challenging problems in computational biology. The interest is not just finding the final fold but of finding the major states that a protein folds into. Because the computational power necessary to calculate a realistic all-atom simulation is far from feasible, scientists aim to simplify the folding process by identifying the major states.

By calculating the characteristics of a protein over time while folding and then clustering these characteristics, it is possible to confirm existing states as well as identify new ones.

Similar to microarray experiments a matrix of data is the result of calculating the characteristics of a protein over time. Also similar to microarray data a clustering technique needs to be applied in order to extract the useful information.

In our research we have addressed the problem of visualization of the above types of data. The motivation for this research is part of our collaboration with two scientists of the Computational Biology Group at the IBM T. J. Watson Research Center in New

York. Dr. Laxmi Parida and Dr. Ruhong Zhou of this center are studying the effects of the β -Hairpin protein folding over time. Comparable to microarray data, clustering of the protein's characteristics proved to be a useful first step. With the very large data set that resulted there was a need for an appropriate visualization tool. We carried out a comprehensive survey of scientific visualization tools available in the public domain. However a single tool that provides all the visualization facilities required for this purpose could not be found. Hence it was decided that we would develop our own customized visualization software.

1.6 Objectives of this Research

The main objectives of the research are:

1. Carry out a comprehensive study of visualization techniques and tools suitable for protein state data
2. Develop a customized visualization software with the following capabilities
 - Provide a visual global view of the data
 - Enable visual identification of major data clusters
 - Enable visualization of relative growth of data clusters over time

1.7 Overview of Thesis

Chapter 2 provides a comprehensive survey of the current state of the art in clustering and visualization techniques used for gene expression data. In Chapter 3 we

describe some details of the protein folding experiments carried out by Dr. Parida and Dr. Zhou and their technique of using patterned clusters for gaining insight into the different states of a protein during the folding operation. In this chapter we also bring out the need for a customized visualization tool. In Chapter 4 we describe the new visualization techniques that we have created to visualize patterned clusters through the use of an animated terrain. **Protein Terrain Analyzer** or **PROTERAN** for short uses mountains that grow to reflect the evolution of major states over time. **PROTERAN** is a novel approach to visualizing patterned clusters and was successfully proven on the β -hairpin protein data provided by Dr. Parida and Dr. Zhou. Chapter 5 outlines the results of **PROTERAN** for Dr. Parida and Dr. Zhou as well as the future work to be done. Finally Appendix A gives a list of distance metrics used to compare genes for clustering while Appendix B provides the mathematics necessary to perform Principal Component Analysis, a common clustering technique.

CHAPTER 2 - Clustering and Visualization Techniques

Clustering is the process of organizing objects into groups whose members are similar in some way [10]. In gene expression, genes of unknown function are clustered with genes of known function in an attempt to learn more about the former⁴. This then poses the problem of how to decide how similar two genes are. In order to determine the similarity of two genes, it is necessary to adopt a mathematical description of similarity [11]. Before a mathematical description can be formulated, the exact data that characterizes a gene must be examined.

2.1 Expression Matrix

The data obtained from a microarray consists of a matrix of fluorescence intensity values for N genes by M experiments. These values are obtained by taking the ratio of C5 (red dye) to C3 (green dye) (*cf.* Table 2-1).

Table 2-1: Expression Matrix

	Experiment 1	Experiment 2	Experiment M
Gene 1	$C5_{11}/C3_{11}$	$C5_{12}/C3_{12}$	$C5_{1M}/C3_{1M}$
Gene 2	$C5_{21}/C3_{21}$	$C5_{22}/C3_{22}$	$C5_{2M}/C3_{2M}$
.
.
.
Gene N	$C5_{N1}/C3_{N1}$	$C5_{N2}/C3_{N2}$	$C5_{NM}/C3_{NM}$

⁴ Though we present clustering and visualization for gene expression data the same techniques are applicable to protein state characteristics.

Therefore a gene in which the sample is overexpressed (i.e. sample is greater than control) would have a fluorescence value greater than one ($C5 > C3$). A gene that is underexpressed ($C3 > C5$) would have a value of less than one and greater than zero. This however does not lend itself to a well dispersed range. Overexpressed genes range between one and infinity while underexpressed genes range only between zero and one. Therefore by taking the logarithmic of the ratio, overexpressed genes will map to positive values and underexpressed genes to negative values. Thus each value in the matrix can now be replaced with $\log_2(C5_{ij}/C3_{ij})$ where i ($0 < i < N$) represents the gene and j ($0 < j < M$) represents the experiment [11].

2.1.1 Data Adjustments

Prior to analysis, different types of adjustments can be performed on gene expression data. Some examples are:

1. Adding random values for missing data
2. Subtracting the mean from each element
3. Removing expressions with little change

2.2 Similarity Metrics

A look at the matrix defines each gene as a vector of experiments or each experiment as a vector of genes. All clustering algorithms use a similarity measurement between vectors to compare the genes or experiments. Appendix A gives a list of distance metrics used along with the applicable formulae.

2.3 Clustering Methods

There are three ways of clustering microarray data:

1. Clustering genes (i.e. rows in the expression matrix)
2. Clustering experiments (i.e. experiments in the expression matrix)
3. Cluster both genes and experiments (biclustering)

The goal is to use clustering algorithms to define clusters that minimize intra-cluster variability while maximizing inter-cluster distances using a distance metric [12].

Clustering algorithms can be divided into two types of methods:

1. Supervised
2. Unsupervised

In supervised clustering, vectors are classified with respect to known reference vectors while in unsupervised clustering no predefined reference vectors are used [11].

2.4 Hierarchical Clustering

Hierarchical clustering is currently the most common technique used to cluster gene expression data. Michael Eisen was the first to cluster gene expression data in 1998 by using hierarchical clustering on the *Saccharomyces Cerevisiae* (Baker's yeast) [13]. This algorithm falls under an unsupervised clustering technique because there is little a priori knowledge of the complete repertoire of expected genes [11].

2.4.1 Algorithm

The hierarchical clustering algorithm constructs a tree of nested clusters based on proximity information. There are two different approaches for the construction of this tree.

1. Agglomerative: Start with all n elements each representing a cluster and merge the most similar elements until only one cluster is left.
2. Divisive: Start with 1 cluster, which contains all the elements until n clusters remain.

Agglomerative hierarchical clustering is preferred because its running time is $O(n^2)$ whereas divisive is $O(2^n)$ [12]. Since agglomerative is the most widely used it is the method we discuss further in detail.

The agglomerative approach is a five-step process.

1. Compare all elements (genes) to each other and create an upper diagonal matrix with all the distances. Each of these elements represents a cluster.
2. Find the two closest clusters.
3. Merge these two closest clusters together. Redo the distance matrix taking into account these two clusters as one bigger cluster.
4. Repeat steps 2 and 3 until only one cluster is left

Figure 2-1 illustrates these steps.

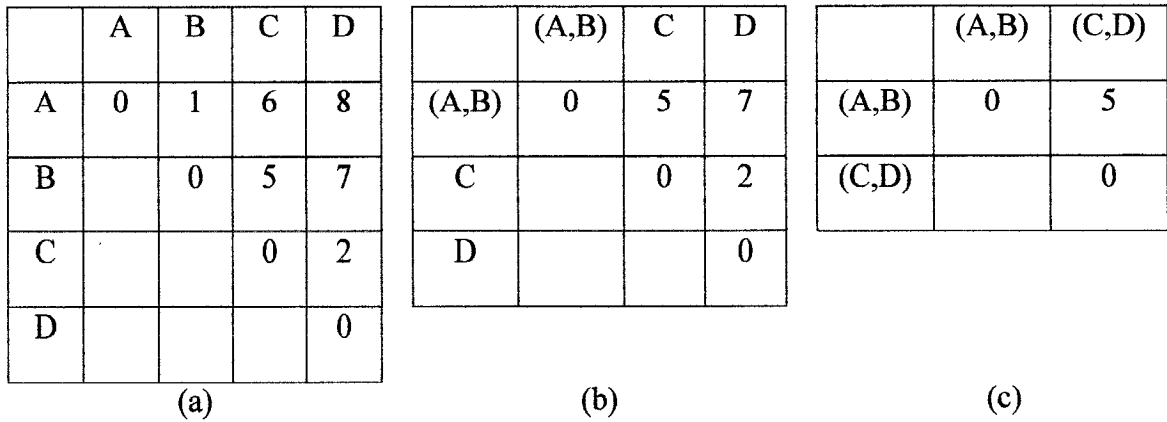


Figure 2-1: Steps 1-4 of Agglomerative Hierarchical Clustering using single linkage.

Computing the distance matrix in step one is simple using a defined similarity metric. The difficulty arises in step 2 when we need to compare distances between clusters that contain more than one element. A variety of methods exist to compute distances when dealing with clusters of more than one element.

1. **Single Linkage Clustering:** Distance between clusters X and Y is the minimum of all pairwise distances between items contained in X and Y
2. **Complete Linkage Clustering:** Distance between clusters X and Y is the maximum of all pairwise distances between items contained in X and Y
3. **Average Linkage Clustering:** Distance between two items X and Y is the mean of all pairwise distances between items contained in X and Y

There are many variations to the above methods with the main one being the use of weights, i.e. clusters that have a lot of elements are weighted more than clusters with fewer elements.

2.4.2 Visualization

The result of the agglomerative hierarchical algorithm is to visualize as a "dendrogram". A dendrogram is a binary tree with a distinguished root, which has all the data items at the leaves. Figure 2-2 depicts a dendrogram created using the results from Figure 2-1.

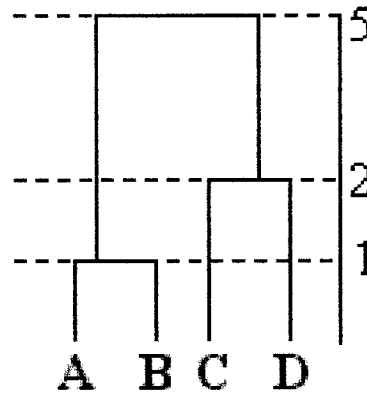


Figure 2-2: Dendrogram 1

For any dendrogram of n elements there are 2^{n-1} ways to structure the tree. In Figure 2-1 there are 4 elements and thus 2^{4-1} or 8 ways to structure the dendrogram. Figures 2-3 and 2-4 are just two other ways.

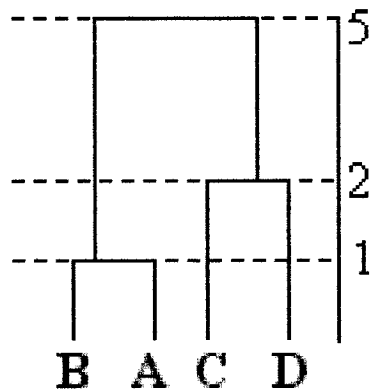


Figure 2-3: Dendrogram 2

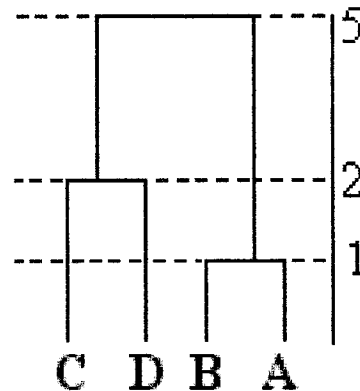


Figure 2-4: Dendrogram 3

This visualization is then coupled with the original expression matrix. Each value in the microarray is colored on the basis of its fluorescence value and re-ordered by the dendrogram. Figure 2-5 shows an image of a dendrogram and the re-ordered expression matrix.

To compute an optimal linear ordering is impractical but through the use of weights such as average expression levels, a satisfactory arrangement can be achieved.

2.4.3 Advantages

This method is very familiar to biologists through its application in sequence and phylogenetic analysis and is probably the reason it is the most popular way to cluster and visualize microarray data. Furthermore besides choosing the similarity metric and the type of method to compare clusters, no further parameters are required. The result is a reordering of the initial expression matrix in which similar genes are placed beside each other.

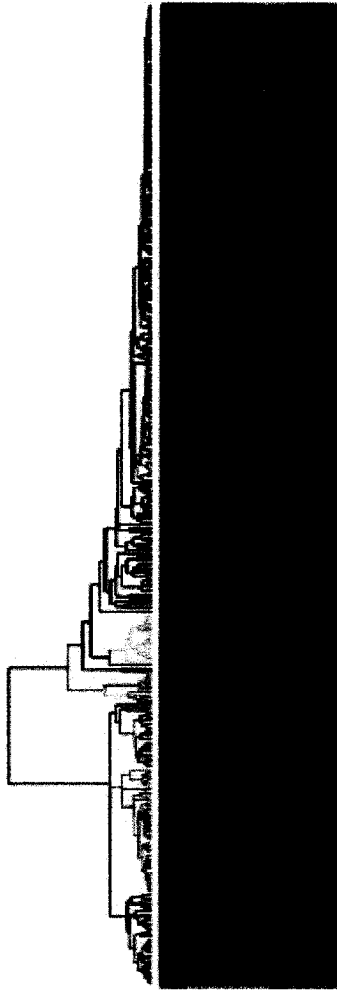


Figure 2-5: Dendrogram and reordered expression matrix⁵

2.4.4 Disadvantages

Despite being the most popular technique, hierarchical clustering and its corresponding visual, the dendrogram have many shortcomings. Large datasets require large amounts of memory and fast CPUs to store and calculate the distance matrix. Another disadvantage is the algorithm just rearranges the dataset leaving the user to find the important clusters (subtrees). In addition hierarchical clustering has been noted by

⁵ <http://www.biostat.wisc.edu/bmi576/fall-2002/lecture15.pdf>

statisticians to suffer from a lack of robustness, nonuniqueness and inversion of problems that complicate interpretation of the hierarchy. Furthermore the deterministic nature of hierarchical clustering can cause points to be grouped based on local decisions with no opportunity to reevaluate the clustering [14].

Despite being very useful in visualizing the cluster decomposition the dendrogram has a significant disadvantage, that of cluster validity. A dendrogram will always suggest $n-1$ different clusters where n is the number of elements to be clustered, regardless of the specific data characteristics [15]. Finally dendrograms only represent hierarchically organized data and do not scale for large amounts of data.

2.5 Self Organizing Maps (SOMs)

Developed by T. Kohonen (1997) and first used on gene expression by P. Tamayo (1998) *et al*, Self-Organizing Maps (SOMS) have proved to be very useful in clustering microarray data [14]. Like Hierarchical Clustering, SOMs is also an unsupervised clustering technique [7], but unlike Hierarchical Clustering, it enables the user to impose partial structure on the clusters [14].

2.5.1 Algorithm

SOMs is the most popular artificial neural network algorithm in the unsupervised learning category [16]. SOMs assume the number of clusters is known and is chosen by the user [9]. The user specifies a grid of “nodes” which represent the clusters, such as an X by Y grid for X times Y clusters. Each of these nodes is then mapped to M -dimensional space where M represents the number of experiments in the microarray. The

initial mapping of the nodes is random and eventually adjusted. Randomly, a vector v from the expression matrix is picked and compared to each of the nodes using one of the similarity metrics discussed in Appendix A. The node's values are adjusted to move closer to this vector v , with the closest node moving the most [16]. This is repeated until all the nodes move less than a certain distance (i.e. nodes have settled) or a certain number of iterations. Each vector in the expression matrix is then clustered by assigning it to the closest node. Figure 2-6 illustrates the steps in a SOMs clustering algorithm.

Besides selecting the number of clusters and the distance metric used, the user must also choose the type of learning function to adjust the nodes with. Two common types are:

1. Neighborhood Function:

All nodes outside a distance of node n , where n is the node closest to vector v , are not modified. All nodes within this distance are modified by $\alpha(i)$, which is the learning rate and decreases with the number of iterations denoted as i . Tamayo *et al.* in their program GeneCluster use the learning function $\alpha(i) = 0.02T/(T + 100i)$ where T is the maximum number of iterations [9].

2. Gaussian Function:

In this approach all nodes are modified but nodes farther away move less. The modification is by the following learning rate $\alpha(i) \cdot \exp(-d(n,n_p)/2\sigma^2(i))$, where $\alpha(i)$ and $\sigma(i)$ decrease with i . This method is more accurate but computationally more expensive.

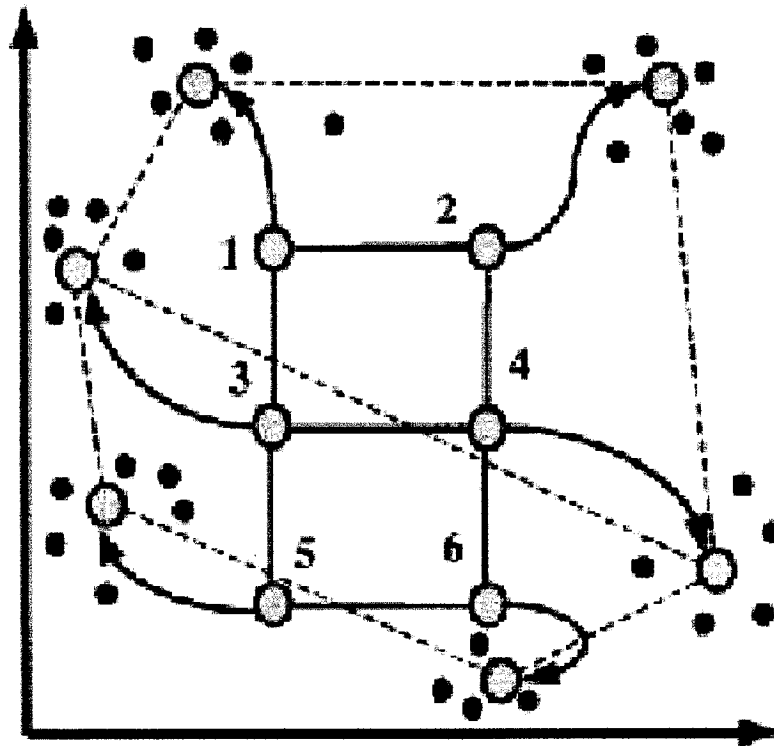


Figure 2-6: Principle of SOMs. Hypothetical trajectories of six nodes as they migrate to fit data during successive iterations of the SOMs algorithm are shown. Data points are represented by black dots, nodes by large circles and trajectories by arrows [14].

2.5.2 Visualization

The algorithm partitions the data into a distinct number of convex Voronoi regions which are specified by the user at the beginning. The result is a group of smaller subsets of the expression matrix in which clusters closer to each other resemble each other more than clusters farther away.

Because the data is partitioned into smaller subsets, there are many ways to visualize the data. One technique is to use the hierarchical clustering method on the subsets and visualize using the dendrogram (*cf.* Figure 2-7) [17].

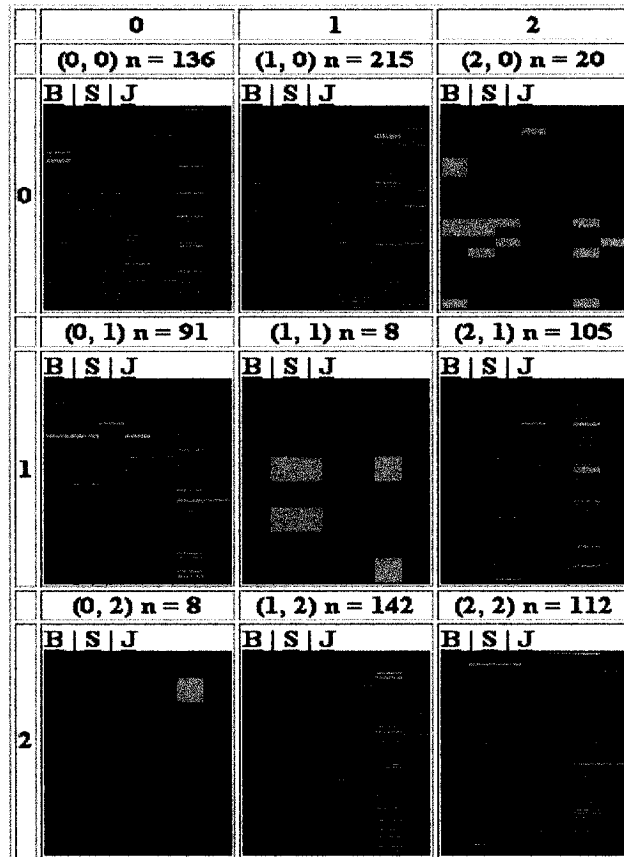


Figure 2-7: Hierarchical Clustering of SOM subsets⁶

Another method is that of plotting each gene's path on a graph where the x-axis represents the time points and the y-axis represents the expression level. One problem is that too many genes in the cluster clutter the visual and it becomes hard to tell genes apart. A solution to this is to use error bars indicating the maximum and minimum variation from the average expression levels (*cf.* Figure 2-8) [14].

⁶ <http://smd.stanford.edu/help/clustering.shtml>

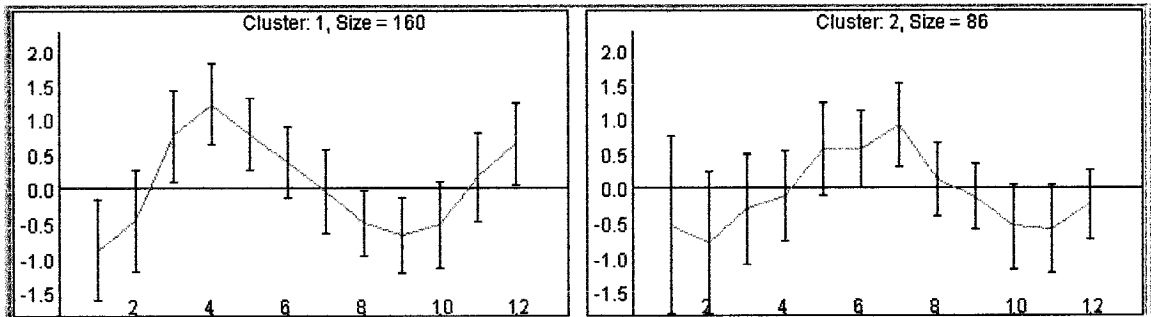


Figure 2-8: Error bar representation of gene expression data⁷.

Besides these techniques the most popular way to visualize the SOM is by the unified-distance matrix or U-Matrix for short (*cf.* Figure 2-9).

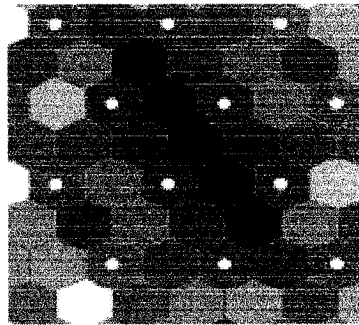


Figure 2-9: U-matrix representation of the SOM⁸

The U-matrix allows easy detection of clusters of nodes by visualizing distances between the neurons [18, 19]. Neurons are colored on the basis of their similarity to other adjacent neurons. The result is a map with an extra element between each neuron depicting the distance between adjacent nodes. One common way of coloring the nodes is with gray levels. A node closer to another node is shaded a lighter gray than a node

⁷ http://www.cs.tau.ac.il/~rshamir/expander/EXPANDER_Images.html

⁸ <http://www.scs.org/scsarchive/getDoc.cfm?id=2363>

farther away. This value is determined by creating a matrix of distances between adjacent nodes using the same distance metric as used in the SOM algorithm and normalized so that the farthest two nodes are black and the closest nodes are white (cf. Figure 2-10).

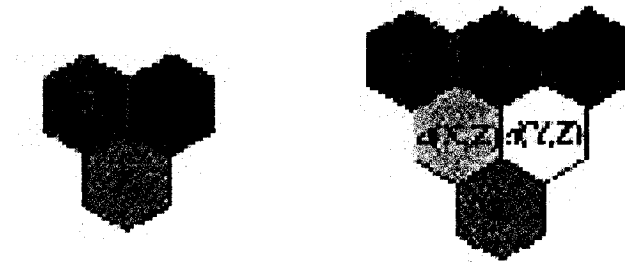


Figure 2-10: Neurons in the two-dimensional graph are colored according to their similarities to adjacent neurons. The result of the U-matrix procedure is an extra element between the neurons. X & Y are similar to each other while Z has a distinct reference vector.

2.5.3 Advantages

Unlike hierarchical clustering, SOMs enables the user to impose partial structure on the clusters and is scalable to large datasets [14]. Also the SOM is able to perform all this with moderate memory consumption and has a running time of only $O(n)$ [7]. Furthermore SOMs can implement fuzzy clusters which allow gene expression vectors to be in more than one cluster.

Because SOMs partitions the data into smaller subsets almost any visual technique can be used on the smaller data. This flexibility cannot be found with Hierarchical Clustering and allows the user to get different perspectives on the same data with different visual techniques.

Finally SOMs can be used on partial data or complete data, another quality that cannot be found with hierarchical clustering.

2.5.4 Disadvantages

Although SOMs allow the user to impose structure on the clusters this requires knowledge of how many clusters there are beforehand. Besides specifying the dimensions of X and Y (number of clusters) many other parameters have to be specified such as:

1. Number of iterations
2. Initial Learning rate α
3. The neighborhood radius
4. Type of neighborhood function
5. Type of vector initialization prior to training
6. Topology of the map

2.6 K-Means

K-means is another algorithm used to cluster gene expression data. Like the SOM, k-means is an unsupervised algorithm that partitions the data. Many variations exist; the following version is the approach by MacQueen (1965) [21].

2.6.1 Algorithm

For a given n , the number of possible partitions is definite but extremely large. Because it is not viable to check every possible partition a cost function is used to find an optimal one. A widely used cost function is defined below [22].

$$E = \sum_{l=1}^k \sum_{i=1}^n y_{i,l} d(X_i, Q_l) \quad (2.1)$$

Where:

k = number of clusters

n = number of vectors

Q_l = mean of cluster l

X_i = the input vector

$y_{i,l} = 1$ if X_i is in cluster l , 0 if not.

d = similarity measure

and: $0 \leq l \leq k$, $0 \leq i \leq n$

The goal of k-means is to minimize this cost function through the following algorithm.

1. Selection of initial k
2. Calculation of similarity between an object and the centroid (Q) of a cluster
3. Allocation of the object to the cluster whose centroid is closest to the object
4. Recalculation of the centroid for which the object was moved from as well as the centroid for which the object was moved to

Steps 2-3 are repeatedly performed until the algorithm converges [22].

Initially k-means starts with k random centroids. Each input data is compared to all centroids and placed in the cluster that it is closest to (*cf.* Figure 2-11(a)). All centroids are then recalculated by taking the average of all the vectors in the respective cluster (*cf.* Figure 2-11(b)). Each input data is then compared to each centroid and then again re-clustered (*cf.* Figure 2-11(c)). This is repeated until no more data points are moved.

One notable variation to the algorithm is that of fuzzy k-means in which one vector can belong to more than one cluster.

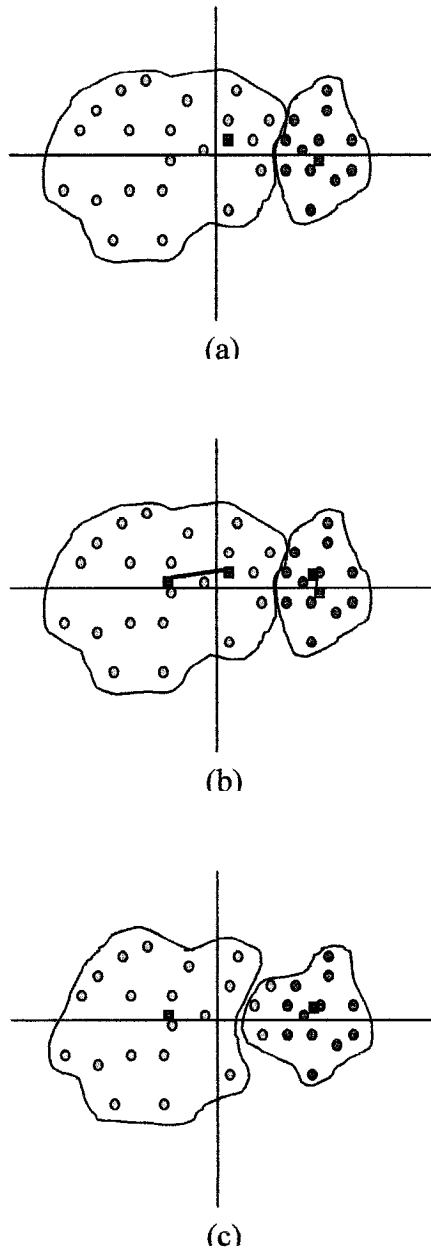


Figure 2-11: Example of K-means, $k = 2$.

- (a) Input data is compared to all centroids & placed in closest cluster.
- (b) Centroids recalculated
- (c) Input data is again compared to all centroids & placed in closest cluster.

2.6.2 Visualization

No specific visualization technique has been developed for k-means clustering. This is because like SOMs, it partitions the data into k distinct clusters and on each of these clusters almost any applicable visualization technique can be used.

2.6.3 Advantages

K-means clustering method is comparable to that of SOMs and thus holds many of the same properties as SOMs. It is best suited for data mining because it is efficient in processing large amounts of data $O(kmn)$ where [22].

- m = vector attributes
- k = number of clusters
- n = number of vectors
- t = number of iterations

Also since there is no similarity matrix to calculate it requires only moderate memory requirements $O(n)$ [7].

Unlike SOMs, only the number of clusters and the number of iterations to prevent infinite calculations are needed to cluster the data.

Finally there is the possibility of creating fuzzy clusters. This has been shown to be a better model for the regulatory system where one gene affects more than one gene [7].

2.6.4 Disadvantages

The number of clusters has to be specified in advance. If k equals the right number then k -means algorithm will cluster the data correctly, if not we could end up with an incorrect clustering result as shown in Figure 2-12 [23].

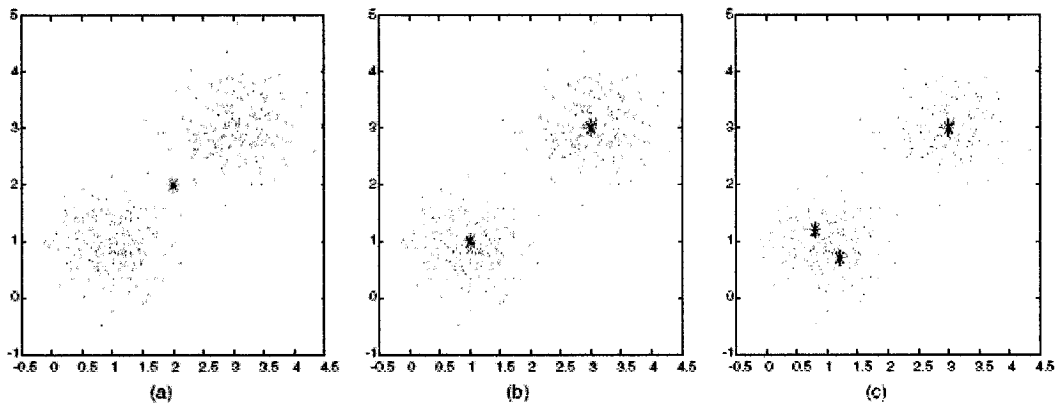


Figure 2-12: The results of the k -means algorithm with different k -values. (a) $k = 1$; (b) $k = 2$; (c) $k = 3$. Only $k = 2$ clusters the data properly [23].

Also it is difficult to discover clusters with non-convex shapes because all the k -mean's clusters have convex shapes [22, 23]. Finally there is the possibility of the “dead-unit” problem where if one unit is initialized far away from the input data then it immediately becomes dead without a learning chance.

2.7 Principal Component Analysis (PCA)

Microarray data with N genes and M experiments can be seen as a mapping into a M -dimensional space. Visual interpretation restricts the number of dimensions to a maximum of 3 (x , y and z). The problem is how to map from this M -dimension to three dimensional space in order to enable visualization.

Principal Component Analysis (PCA) is a mathematical process that reduces the M variables into M principal components called factors. Each factor is ordered in importance where the first factor explains the most about the differences between the data, the second factor as much as what the first factor could not and so on. By finding the principal components of a microarray experiment the first 3 factors that describe the data as much as possible could be displayed in a 3D environment where x , y , z represent factors 1, 2 and 3 respectively. An example of condensing the data from two dimensions to one dimension is shown in Figure 2-13. It must be noted that the M factors are not just a re-ordering of the M experiments but M principal components that account for as much of the variance in the original data.

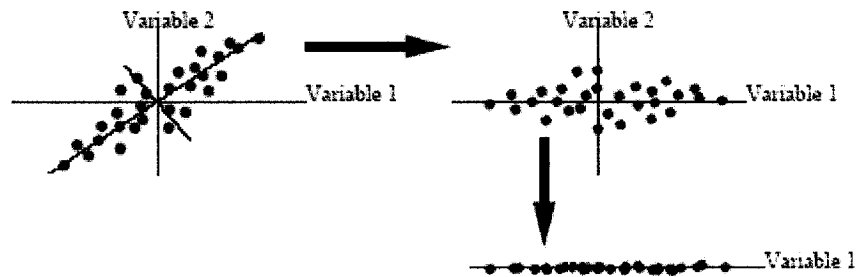


Figure 2-13: Principal component analysis

2.7.1 Algorithm

For mathematical concepts in PCA please refer to Appendix B.

1. Process the data:

One method is to subtract the mean from each data dimension.

This is an optional step and is left to the discretion of the user.

2. Calculate the covariance matrix:

Create an $M \times M$ covariance matrix from the $N \times M$ data matrix.

3. Calculate the eigenvectors and eigenvalues:

Using the $M \times M$ covariance matrix calculate the m eigenvectors and their respective eigenvalues. By finding these eigenvectors, we are able to extract the lines that characterize the data.

4. Form the feature vector:

The eigenvector with the highest eigenvalue is the first principal component while the second highest eigenvalue is the second principal component and so on. Thus to reduce the dimensionality the components that are least significant are thrown away and the components kept are stored in a matrix called a feature vector where the eigenvalues represent the columns.

$$\text{FeatureVector} = (eig_1 \ eig_2 \ eig_3 \dots \ eig_N)$$

5. Create the New Data Set:

To get the final data the transpose of the feature vector is multiplied by the transpose of the original data.

$$\text{FinalData} = \text{FeatureVector}^T \times \text{Original Data Matrix}^T$$

The final data is now expressed as an $N \times P$ matrix where N is the number of genes and P is the number of principal components.

2.7.2 Visualization

The top three principal components are usually chosen in order for the data to be compressed to an $N \times 3$ matrix. Each gene can then be plotted in 3-dimensional space where principal component 1, 2 and 3 represent the x, y and z axis respectively. Figure 2-14 shows an example visualization of PCA on gene expression:

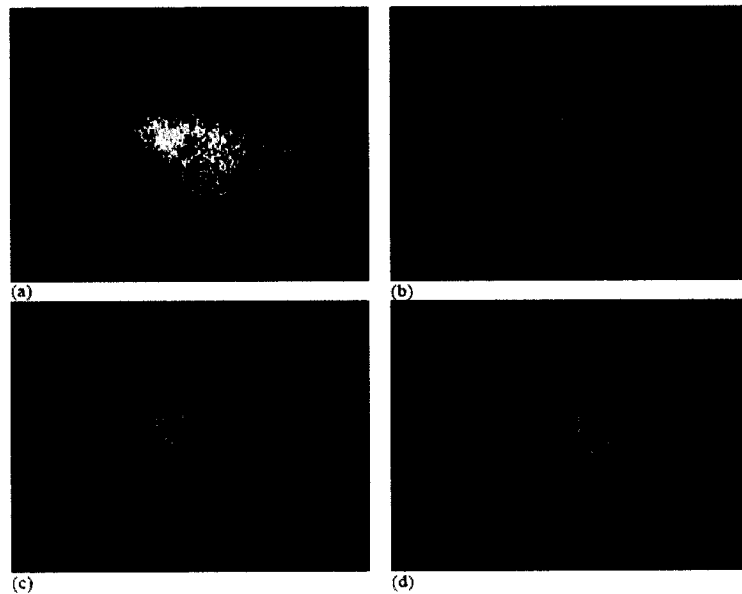


Figure 2-14: Visualization of PCA on gene expression [7].

The net result is points close to each other in the Principal Component Space have the same basic patterns and thus have similar gene expression.

2.7.3 Advantages

Unlike the previous methods no parameters have to be specified beforehand. Also besides being deterministic, PCA removes the noise from the data [7].

Until now all the visualization techniques used on the clustering algorithms discussed have been 2D, whereas PCA inherently proposes 3D. Finally the 3D world is more flexible than the 2D world and users often find it more intuitive [24].

2.7.4 Disadvantages

Since there are no parameters to specify, the user has little control on the outcome of the data. Furthermore PCA has a computational complexity of $O(N^3)$. While all the earlier methods can work with both gene expression data as well as distance matrix data, PCA is only applicable to gene expression data. It is not applicable when only a distance matrix of genes is provided as data [24].

2.8 Biclustering

Typical clustering methods look at either clustering gene expression levels in all experiments (rows in the expression matrix) or expression levels of all genes considered in each experiment (columns in the expression matrix) [25]. Any formula of equally weighing all conditions or genes will lead to the discovery of a similarity group at the expense of obscuring other similarity groups. Biclustering is a novel approach first introduced on gene expression data by Cheng and Church in 2000 to overcome the problem of information lost during the use of oversimplified clustering techniques [21]. This is done by simultaneously clustering both rows and column sets to obtain a subset of the data matrix with a high similarity score. With this, the involvement of a gene or a condition in multiple pathways can be determined. Other terms to describe biclustering are “direct clustering” and “box clustering” [21].

2.8.1 Algorithm

The algorithm for biclustering measures the coherence of the genes and conditions by calculating the mean squared residue. The mean squared residue of an element a_{ij} is defined as

$$a_{ij} - a_{i\cdot} - a_{\cdot j} + a_{\cdot\cdot}$$

where

$a_{i\cdot}$ = Mean of the i th row in the bicluster

$a_{\cdot j}$ = Mean of the j th column in the bicluster

$a_{\cdot\cdot}$ = Mean of all elements in the bicluster

The mean squared residue is the variance of the set of all elements in the bicluster, plus the mean row variance and the mean column variance. The objective is to find large clusters with low mean squared residue. In addition to finding the low mean square residue a reasonably large row variance is also specified. The reason for this is that the mean squared residue only shows that genes and conditions fluctuate in unison and therefore this includes trivial biclusters, those with little or no fluctuation at all [26]. The most interesting biclusters are those that have low mean square residue and show similar up-regulation and down-regulation under a set of conditions [26]. Figure 2-15 shows how biclustering can be more advantageous over traditional clustering techniques.

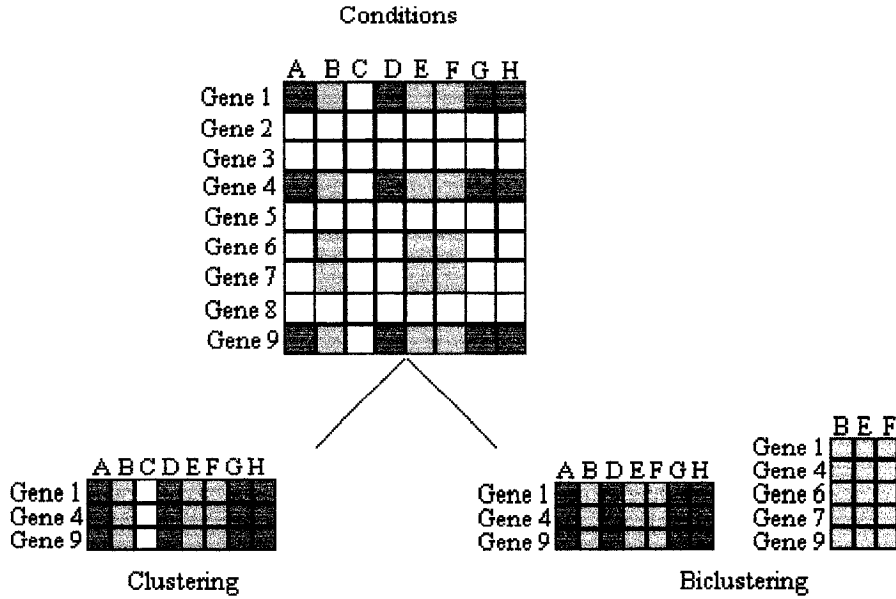


Figure 2-15: Biclustering versus traditional clustering

Partitioning the rows and columns to find the optimal biclusters has been proven to be an NP-hard problem [21, 25, 26]. Therefore heuristic algorithms have been designed to find acceptable biclusters. The following is a brute force algorithm by Cheng and Church. It has a running time of $O((N+M) \times NM)$, where N and M are the row and column sizes of the expression matrix. For more efficient algorithms, see “Biclustering of Expression Data” by Cheng and Church [21].

The following is defined for the mean squared residue score of a matrix.

$$\frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} (a_{ij} - a_{i\cdot} - a_{\cdot j} + a_{\cdot\cdot})^2 \quad (2.2)$$

Algorithm 0 (Brute-Force Deletion and Addition)

Input:

A , a matrix of real numbers, and $\delta \geq 0$, the maximum acceptable mean squared residue score.

Output:

A_{ij} , a δ -bicluster that is a submatrix of A with rows set I and column set J , with a score no larger than δ .

Initialization:

I and J are initialized to the gene and condition sets in the data and

$$A_{IJ} = A.$$

Iteration:

1. Compute the score H for each possible row/column addition/deletion and choose the action that decreases H the most. If no action will decrease H , or if $H \leq \delta$, return A_{IJ} .

2.8.2 Observations

By taking into account both the rows and columns of the expression matrix, biclustering differs from previous clustering algorithms discussed. As we will see in Chapter 3 the notion of biclustering has similarities with the clustering algorithm used by Dr. Parida and Dr. Zhou. Although their algorithm is a combinatorial “optimization” problem with an output sensitive algorithm (work proportional to the size of the output) the end result is similar in that clusters contain both rows and columns.

2.9 Available Software

Many of the clustering and visualization techniques discussed in this chapter are available in a wide variety of software packages, both in the public domain and from

vendors. Similarly the imaging software for microarrays in Chapter 1 is also widely available. Table 2-2 lists some of these software packages.

Table 2-2: Some Image analysis and Data Mining Software Packages

Software Name	Description	Available at
F-Scan	Quantification and analysis of fluorescently probed microarrays; scatterplots; multiple image comparison.	http://abs.cit.nih.gov/fscan/
TIGR SpotFinder	Spot identification.	http://www.tigr.org/software/
Cluster	Hierarchical clustering, K means clustering Self-Organizing Map (SOM), PCA	http://rana.lbl.gov/EisenSoftware.htm
Genesis	A Java suite containing various tools such as filters, normalization, visualization tools, common clustering algorithms, SOM, k-means, PCA,	http://genome.tugraz.at/Software/GenesisCenter.html
J-Express Pro 2.0	Hierarchical clustering, K-means, Principal Component Analysis, Self-organizing maps, Profile similarity search, Normalization and filtering, Raw data import, Project organization	http://www.molmine.com/frameset/frm_jexpress.htm
TreeView	Cluster output visualization	http://rana.lbl.gov/EisenSoftware.htm

In most situations, when the visualization requirements are standard it is possible to use one or more of the packages available.

CHAPTER 3 - Protein Folding Trajectory Analysis

The way a protein folds and its final native structure determine the function of that protein. Since genes are the genetic code for producing proteins, finding the function of the protein is essentially finding the function of the gene.

Besides obtaining the final folded state, it is also important to understand the kinetic process of folding [27]. Determining how a protein folds is one of the hardest problems in molecular biology. Certain proteins called fast-folders can go from an unfolded to a folded state in a matter of microseconds or milliseconds. With this short time frame, along with experimental and hardware limitations, mapping the full folding process has proven to be a difficult task.

An all-atom realistic recreation of a protein's kinetic folding process is well beyond the reach of today's technology. Therefore the folding mechanism is usually characterized by calculating the thermal dynamics of the folding, i.e., the free energy landscape. The folding free energy landscape is often characterized by contour maps versus the so-called reaction coordinates (reduced coordinates to describe protein structures due to too many degree-of-freedom). Below is a list of some reaction coordinates:

1. Fraction of native contacts
2. Radius of gyration of the entire protein
3. RMSD from the native structure
4. Number of beta-strand Hydrogen bonds
5. Number of alpha-helix turns
6. Hydrophobic core radius of gyration

7. Principal Components

Through the course of folding, a protein goes through certain shapes or states before settling on a final fold. In order to understand this process an obvious step is to find the intermediate states that occur through the course of folding (*cf.* Figure 3-1). Through the use of a combinatorial pattern discovery algorithm, Dr. Laxmi Parida and Dr. Ruhong Zhou at the IBM T J Watson Research Center are working on the discovery of existing states as well as new ones.

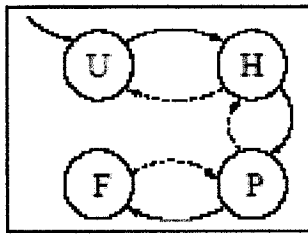


Figure 3-1: A hypothetical state diagram of a folding protein.

The combinatorial technique involves clustering of the protein's reaction coordinates or characteristics in order to find the states. This method is analogous to the clustering of microarray data with the slight exception that the clustering is a matrix of the protein's characteristics whereas microarray data clusters a matrix of gene expression.

3.1 Data

To demonstrate the effectiveness of their technique Dr. Parida and Dr. Zhou tested their algorithm on the β -hairpin, a small protein which has been studied extensively. The protein was solvated and subjected to temperatures that ranged between

270K and 695K. Seven characteristics of the protein were then measured at different time intervals. These time intervals are usually equal, but the time sequence is not necessarily in real time, since these simulations were carried out with the Replica Exchange Method (REM), which is essentially a Monte Carlo method [28]. A list of the seven reaction coordinates are used for this particular β -hairpin:

1. Number of Native beta-strand hydrogen bonds
2. Radius of gyration of the hydrophobic core residues
3. Radius of gyration of entire protein
4. Fraction of native contacts
5. Principal component 1
6. Principal component 2
7. Root mean square deviation (RMSD) from the native structure.

For the details of the reaction coordinates, including their definitions and the way they are calculated, the reader is referred to [28].

Table 3-1 shows a sample data set:

Table 3-1: Sample data of β -hairpin's reaction coordinates over time [27].

J_1 N_{HB}^{β}	J_2 R_g^{conc}	J_3 R_g	J_4 ρ	J_5 $PC-1$	J_6 $PC-2$	J_7 $RMSD$
5.000	5.175	8.653	1.000	-7.819	-34.008	0.000
4.468	5.394	8.425	0.991	-7.908	-35.604	1.575
4.474	5.328	8.361	0.953	-7.972	-35.772	1.595
4.354	5.416	8.471	0.988	-7.899	-36.399	1.379
4.159	5.589	8.379	0.938	-8.171	-34.609	1.439
4.000	5.445	8.418	0.933	-8.724	-35.593	1.626
4.053	5.257	8.298	0.893	-8.373	-35.536	1.708
3.776	5.186	8.381	0.857	-7.777	-35.415	1.624
2.398	5.268	7.795	0.778	-2.749	-26.391	3.726
2.155	5.390	7.816	0.778	-2.277	-27.017	3.672
4.842	6.043	7.312	0.778	2.144	-33.772	5.208
0.000	8.466	10.134	0.249	-24.492	44.625	10.357
0.000	8.303	10.033	0.242	-27.075	43.521	10.163
2.047	5.132	7.628	0.776	-3.238	-24.908	3.927
3.797	5.990	7.514	0.728	-3.084	-30.185	4.838
2.898	5.483	7.775	0.778	-2.888	-26.254	3.904

A total of 19087 different time points were measured giving a matrix consisting of 133609 (19087*7) elements. Furthermore a threshold was added to each characteristic in order to limit the total number of patterns (*cf.* Table 3-2).

Table 3-2: Threshold added to each characteristic.

Characteristic 0	± 0.2
Characteristic 1	± 0.6
Characteristic 2	± 0.35
Characteristic 3	± 0.15
Characteristic 4	± 5.0
Characteristic 5	± 16.5
Characteristic 6	± 1.0

For example row 1 of Table 3-1 values with Table 3-2 thresholds added can vary in between any of the following values as depicted in Table 3-3.

Table 3-3: Results after adding threshold

4.8 to 5.2	4.575 to	8.303 to	0.850 to	-12.819 to	-50.508 to	-1.0 to 1.0
	5.775	9.003	1.150	-2.819	-17.508	

3.2 Patterned Clusters

In order to find the major states, a “patterned cluster” is extracted from the data which is defined as a cluster of some column combinations that satisfy a quorum k, (the specific pattern is present (within tolerance) at least k times).

Each Patterned Cluster is described in two rows. The first row gives the pattern as number of columns, and subsequently each column number and its corresponding value. The second row gives the number of occurrences and the list of occurrences in the original data. For example, a single patterned cluster could be described by Figure 3-2:

2	0	0.1	4	0.23	← Pattern
3	23	26	27		

Figure 3-2: Patterned cluster

Definition of “Pattern”:

A pattern is defined as the total number of columns along with the actual columns and their respective values.

Definition of “Patterned Cluster”:

A patterned cluster is defined as a pattern along with the total number of occurrences and the list of occurrences.

With these definitions, the data in Figure 3-2 would be interpreted as:

- 2 is the number of columns
- 0 and 4 are the columns
- 0.1 and 0.23 are column 0 and column 4 values respectively
- 3 is the number of time occurrences
- 23, 26 and 27 are the time points where the pattern occurs

A further requirement was that patterned clusters that appeared less than a certain chosen amount of time, say k were discarded. The result is that any patterned cluster that appears less than k times is not considered a major state.

After calculating the patterned clusters and removing the clusters with less than k occurrences the net result is a large file. The data file supplied to us was over 500 MB. Table 3-4 is a sample cut out of the patterned cluster file.

Table 3-4: Sample patterned cluster file

2	0 7.335	1 0.735					
1006	59728	87235	94826- 94831	95748- 95752	95761- 95763	...	120424- 120426
2	0 7.335	1 0.736					
1003	59728	87235	94826- 94831	95748- 95752	95761- 95763	...	95769
2	0 7.335	1 0.737					
1012	59728	72071	87235	94826- 94831	95748- 95752	...	95767
2	0 7.335	1 0.738					
1028	59728	72071	87235	94826	94828- 94831	...	95761- 95763
3	0 7.335	4 - 5.881	6 3.292				
1036	59728	72071	87235	94826	94828- 94831	...	95761- 95763
3	0 7.335	4 - 5.881	5 2.214				
1056	59728	72071	87235	94826	94828- 94831	...	95761- 95763
:							
:							
:							
:							
5	2 8.144	3 0.899	4 - 3.855	5 - 33.574	6 3.292		
1089	45533	59728	72071	87235	94826	...	95748- 95752

By analyzing the data in this fashion it is possible to confirm both existing states as well as identify new states. This appears to be an innovative way of understanding the folding process of a protein.

3.2.1 The Need for Visually Analyzing the Patterned Cluster Data

It is quite clear that with the large amount of data contained in the patterned cluster data file, it cannot be easily analyzed by manual study. Memorizing ten patterned

clusters is a difficult task, and with thousands of clusters it is virtually impossible. Therefore obtaining a global view of the data is very important. Furthermore the interaction of the patterned clusters in relation to each other is hard to ascertain when placed one after the other in a data file. The user is now left with trying to memorize both the patterned clusters and their interaction with each other. Finally the patterned clusters are occurrences in time with some patterns occurring before other patterns. This lends itself to a third characteristic of data that needs to be organized by the user, i.e. that of patterns that occur together and those that occur before and after each other. A software tool that enables a graphic visualization of the above dataset would considerably enhance the data analysis process.

3.2.2 Program Requirements

In order to overcome these shortcomings it was necessary to provide a visualization tool for the patterned cluster data. In consultation with Dr. Parida and Dr. Zhou the requirements were formulated as follows:

3.2.2.1 Global View:

Get a global picture (visual) of the data at any time point during the course of folding.

3.2.2.2 Navigation and focus:

Help identify important patterns and patterned clusters and easily navigate through the dataset to focus on any individual patterned cluster.

3.2.2.3 Relative growth rates:

The patterned cluster growth should be animated in order to be able to analyze the relative growth of patterned clusters over time.

3.2.2.4 Detailed Query:

The columns, corresponding values and size of the patterned cluster should be obtainable from the original dataset at any time point.

3.3 The need for a Custom Visualization Tool

Based on the above requirements we analyzed the applicability of the available visualization tools for gene expression datasets. A careful analysis of the existing visualization techniques discussed in Chapter 2 quickly showed they would not be adequate in meeting the requirements set forth in section 3.2.2. Since the clustering method used by Dr. Parida and Dr. Zhou is not hierarchical in nature the dendrogram had to be ruled out. At first glance, the U-Matrix visualization seemed to be a possible solution. By representing each patterned cluster as the neurons, adjacent states could be placed beside each other. The functions to zoom-in, achieve a global view and query the data could also easily be implemented. However animation of the patterned clusters using the U-Matrix was not visually appealing or informative and therefore had to be abandoned. Finally the last technique of using PCA is more of a clustering technique and simple plotting of protein folding states in 3D does not meet the above requirements.

This led to the conclusion that it was necessary to develop a customized interactive visualization tool to specifically address the above requirements. The next

chapter describes the visualization software designed and developed and some results from its application.

CHAPTER 4 - PROTERAN

4.1 Terrain Metaphor

As we concluded in the last chapter, the clustering and analysis needs for the Beta-Hairpin data is different from the usual clustering methods in bioinformatics. Thus a new visualization technique had to be developed. The nature of placing the states on a plane similar to the U-Matrix and the 3D visualization of PCA were both very appealing. Recently the paper “A Gene Expression Map for *Caenorhabditis Elegans*” was published in which clusters of genes were represented as mountains in a terrain [29]. The initial genes were placed in a two-dimensional image using force-directed placement and raised as mountains based on the density of genes in an area. This sparked the idea of using the metaphor of a terrain to achieve the requirements as outlined in section 3.2.2.

This terrain metaphor has proven to be a successful tool in visualizing large amounts of data. Matthew Chalmers in his paper “Using a Landscape Metaphor to Represent a Corpus of Documents” indicated the usefulness of having a consistent ground plane to increase data exploration [30]. By representing each mountain as a patterned cluster that grows over time, the best of both the U-Matrix and PCA visualizations could be combined. Since the mountains are placed along a plane and each represents a state, this is similar to the U-Matrix. Because they are mountains and navigation will occur in 3D the visual is similar to that of PCA’s. Most importantly since the mountain growth can be shown over time, requirement 3.2.2.3 is successfully met. Also by pointing and clicking on a mountain the values of that patterned cluster could be obtained, thus solving requirement 3.2.2.4. Finally the ability to zoom in and out as well as navigate through the terrain would solve requirements 3.2.2.1 and 3.2.2.2.

The final result is the program “**Protein Terrain Analyzer**”, or **PROTERAN** for short. **PROTERAN** is an interactive visualization program to analyze the states of a protein through the animation of a terrain (*cf.* Figure 4-1).

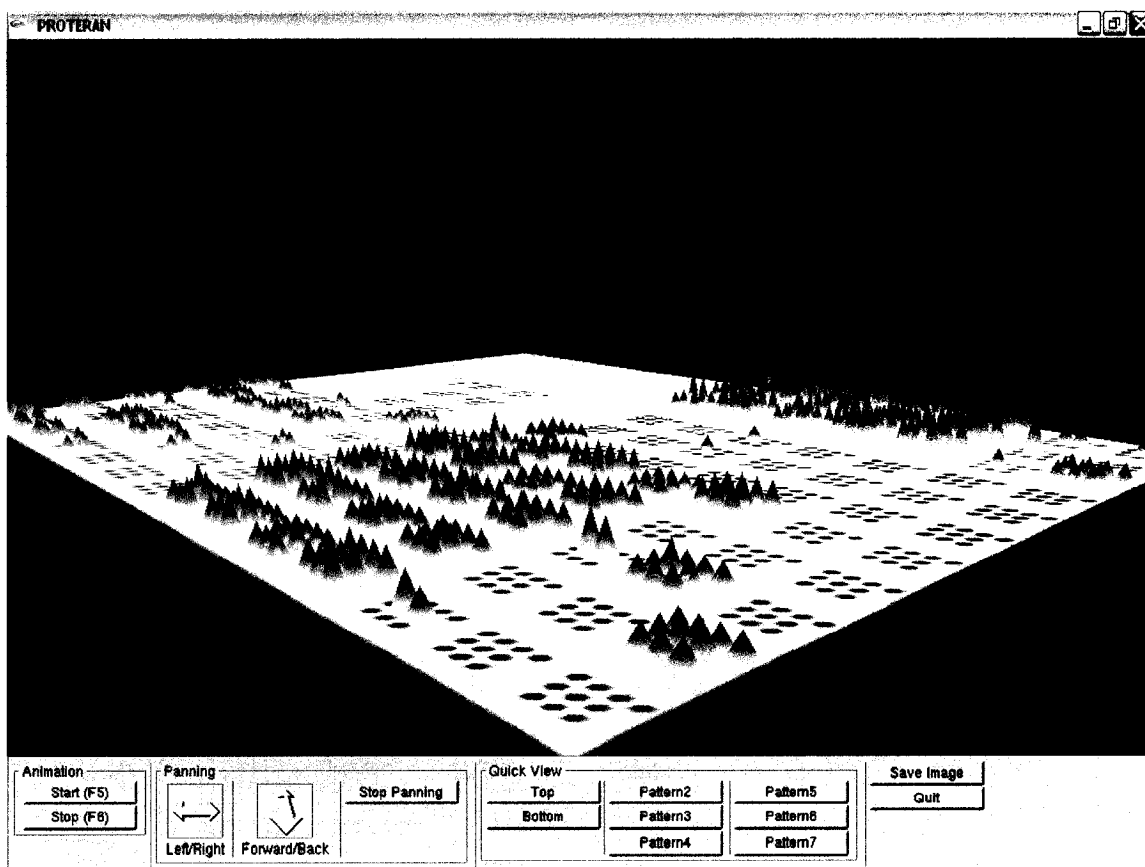


Figure 4-1: PROTERAN: Each mountain represents a patterned cluster.

4.2 Mapping of Patterned Cluster Data into Terrain Geometry

The first step to visualizing the patterned cluster data as a terrain is to be able to differentiate between the patterned clusters. An important design decision was to map patterned clusters as separate mountains and to map the size of the patterned cluster to the height of the mountain. The next design decision was to decide on the layout for the different mountains on the ground. Before we discuss our layout design, we need the following definitions.

Since each patterned cluster can have 2-7 columns, each patterned cluster can be defined by the number of columns.

Definition of “Pattern Type”:

The pattern type is defined by the number of columns in a patterned cluster with the minimum number of columns being two.

With seven characteristics the result is six pattern types. Each of these six pattern types can further be broken down into the number of column combinations.

Definition of “Column Combination”:

The column combination is a unique number that is used to identify a particular combination of columns, generated by merging the columns together.

For example there are 21 different column combinations for type 2 patterns as shown in Table 4-1.

Table 4-1: Patterns of type 2

PATTERNS OF TYPE 2		
Combination ID	Column ID	Column ID
01	0	1
02	0	2
03	0	3
04	0	4
05	0	5
06	0	6
12	1	2
13	1	3
14	1	4
15	1	5
16	1	6
23	2	3
24	2	4
25	2	5
26	2	6
34	3	4
35	3	5
36	3	6
45	4	5
46	4	6
56	5	6

For each pattern type 't' with 'c' columns the number of column combinations can be characterized by the following standard permutation formula:

$$\frac{t!}{(t-r)! * r!} \tag{4.1}$$

Table 4-2 lists the number of column combinations for each pattern type.

Table 4-2: Number of permutations of each pattern type.

Pattern Type	Number of Column Combinations
2	21
3	35
4	35
5	21
6	7
7	1

Thus each Patterned Cluster can now be categorized as a Pattern Type and a Column Combination. For example, Figure 3-2 in section 3.1 can now be classified as:

Pattern Type: 2

Column Combination: 04

With the ability to breakdown the patterned clusters into pattern types and column combinations a method of laying them out as a mountainous terrain is formulated next.

4.2.1 Patterned Cluster Layout

The layout problem can be stated as follows:

For each patterned cluster in the dataset

1. Assign a unique position on the ground
2. Allocate an area surrounding the position for growing the mountains representing the patterned cluster.

We first considered the use of a generic approach such as using the mass-spring framework. The size of a patterned cluster would be the mass and the spring contents would be based on nearness of two patterned clusters. However after extensive consultations with users (Dr. Parida and Dr. Zhou) it was decided that proximity by type

and size was more desirable. Also a fixed ground position for a particular pattern type would provide a visual consistency that was important if the visualization tool were to be used for analysis of different data sets. Hence it was decided that a fixed layout would be designed with built-in flexibility to accommodate different column combinations. Accordingly the layout design was worked out in 3 steps as described below

Step 1:

The first step divides the plane into six blocks, each representing the type of pattern such as shown in Figure 4-2 below.

Pattern 2	Pattern 3	Pattern 4
Pattern 5	Pattern 6	Pattern 7

Figure 4-2: Layout of Pattern Types on the Ground Plane

Step 2:

The second step divides these ground blocks into the required number of column combinations. Since the pattern types do not have an equal number of column combinations the layout was moved around in order to get a more rectangular shape. (*cf.* Figure 4-3)

01	02	03	01234	01235	01236	012	013	014	015	016
04	05	06	01245	01246	01256	023	024	025	026	034
12	13	14	01345	01346	01356	035	036	045	046	056
15	16	23	01456	02345	02346	123	124	125	126	134
24	25	26	02356	02456	03456	135	136	145	146	156
34	35	36	12345	12346	12356	234	235	236	245	246
45	46	56	12456	13456	23456	256	345	346	356	456
						0123	0124	0125	0126	0134
						0135	0136	0145	0146	0156
			012345	012346	012356	0234	0235	0236	0245	0246
	0123456		012456	013456	023456	0256	0345	0346	0356	0456
			123456			1234	1235	1236	1245	1246
						1256	1345	1346	1356	1456
						2345	2346	2356	2456	3456

Figure 4-3: Layout of Column Combinations

This layout not only gives a visually more appealing view of the terrain but also highlights pattern types 6 and 7 because of their separation from other pattern types. This was another requirement set forth by the users, who reasoned that pattern types 6 and 7 are less likely to occur and therefore should be more visible than other pattern types.

Step 3:

A quick analysis of the patterned cluster data revealed there were far too many patterned clusters with little variation in the data. Displaying all of these as mountains would result in too much of visual clutter.

A final requirement was worked out of that m largest patterned clusters for each column combination should be visualized, with m being a user defined number typically around 10 and up to 25. This final requirement further divided each column combination to represent these top m occurring patterned clusters. The most occurring patterned cluster would be placed in the center and the rest in a clockwise fashion radiating outwards. Table 4-3 depicts the outcome for Pattern Type 2 with column combination 01 and $m = 10$.

Table 4-3: Pattern 2 with combination 01

	10 TH Highest Occurrence of combination 01			
	9 TH Highest Occurrence of combination 01	2 ND Highest Occurrence of combination 01	3 RD Highest Occurrence of combination 01	
	8 TH Highest Occurrence of combination 01	Highest Occurrence of combination 01	4 TH Highest Occurrence of combination 01	
	7 TH Highest Occurrence of combination 01	6 TH Highest Occurrence of combination 01	5 TH Highest Occurrence of combination 01	

4.2.2 Animated Terrain Evolution

After deciding on the layout of the mountains the next step was to animate the evolution of the terrain. By animating the terrain, the growth of the patterned cluster through time is simulated. The time proceeds from 0 to the maximum number of experiments with each time unit representing an experiment. At time unit 0 the terrain is completely flat. With each time unit all patterned clusters are checked to see if there was an occurrence. If there is an occurrence, then the height of that respective patterned cluster's mountain is increased. If not, the height remains the same. For this algorithm the data in the original format had to be converted so that it could be rapidly indexed by time points. For very large data sets, this could be time consuming. But since this can be done in a preprocessing stage, it does not in any way affect the real-time performance of this animation.

4.2.3 Interaction Facilities

The user interface has to be simple and intuitive. Realistic terrain rendering was given less importance than the facilities such as being able to navigate freely, visually discriminate between patterned clusters and query any visible mountain for the constituent data or patterned cluster. Accordingly the following rendering and interface design decisions were made.

1. Pseudo coloring of mountains with colors appropriately chosen for easy discrimination from both distant and zoomed-in views
2. For navigation, the "flying through terrain" metaphor was chosen. Using the mouse/keyboard buttons it is possible to virtually fly

through the terrain and “land on any mountain”. Zooming into a specific part of the terrain is automatically supported as part of the flying through interaction. The flying through is supported both for a static terrain (terrain at any time point) or for an evolving terrain.

3. For querying details of any patterned cluster one has just to land. This is done by clicking on any particular mountain during the fly through. The click immediately pops-up an information window which contains all the details of that patterned cluster.
4. Other loading/bookkeeping operations were provided using a simple pull-down menu.

4.3 Programming Tools

PROTERAN was created using the C programming language with Visual C++ 6.0 as the development environment. The basic structure of **PROTERAN** was obtained from the terrain tutorial at <http://www.lighthouse3d.com/opengl/>. The sample code obtained from this site created a terrain from an image file by using the pixel values as the heights of the mountains. Each value was stored in a dynamically allocated two dimensional array representing the layout of the terrain. These values were triangulated and rendered using OpenGL, the most widely used graphics application programming interface (API) in the industry [31].

OpenGL consists of Graphics Library (GL) and Graphics Library Utilities (GLU) functions. GL contains only primitives while the functions of GLU use the functions of GL. OpenGL User Toolkit (GLUT) was also used in the creation of **PROTERAN** to

create windows that are platform-independent, i.e. can be run on both Windows and Macintosh machines [32].

Finally OpenGL User Interface (GLUI), a GLUT based user interface library was used to create the interface of **PROTERAN**. Since it is a wrapper to GLUT it too is portable to any system.

4.4 PROTERAN

The following outlines the functionality of **PROTERAN** in its current version.

4.4.1 Opening Screen

The initial screen offers 3 options (*cf.* Figure 4-4).

4.4.1.1 'Filter File' Button

This button is used when original patterned cluster is input. This is processed and the file "Filtered_File.txt" is created which contains the top 10 patterned clusters of each column combination. The terrain automatically loads after creating the "Filtered_File.txt"

4.4.1.2 'Processed File' Button

This button is used when a processed file is input, i.e. contains only ten patterned clusters of each type of patterned cluster.

4.4.1.3 'Quit' button

Closes **PROTERAN**.

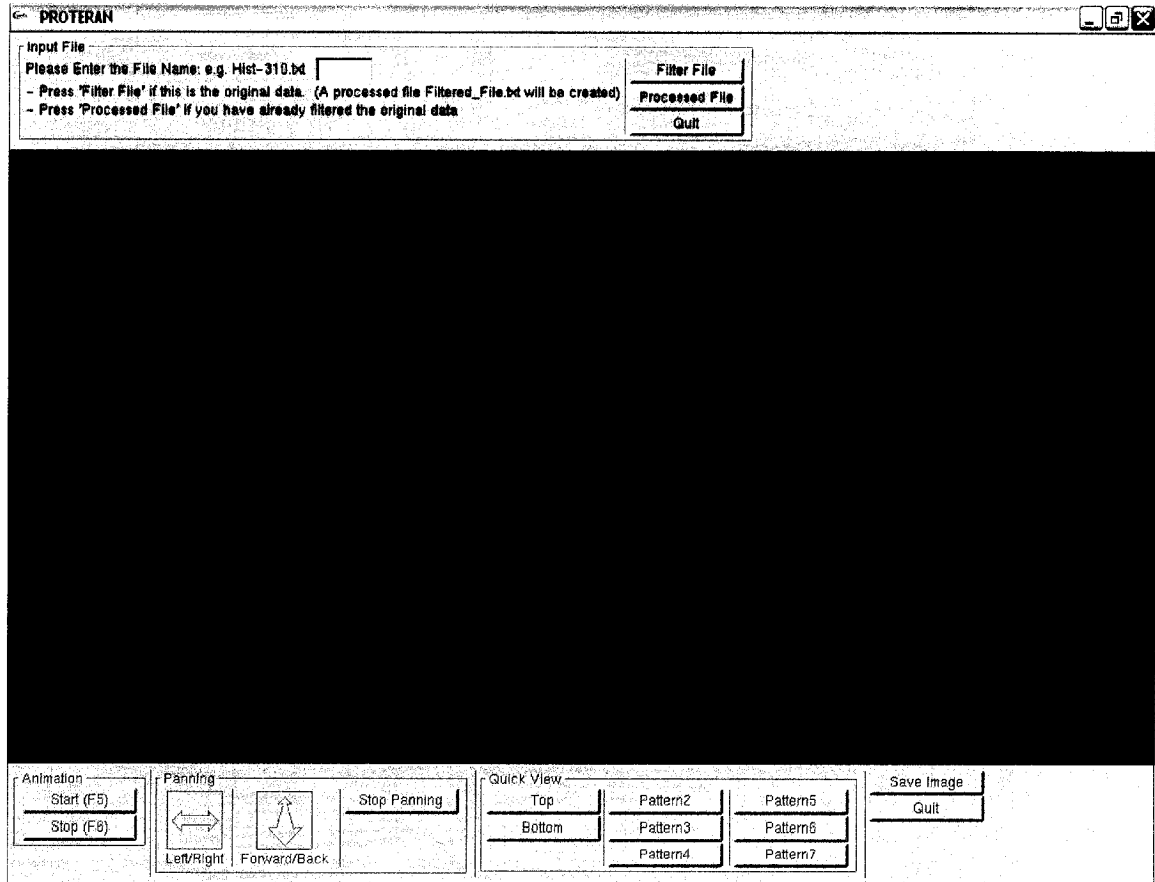


Figure 4-4: Opening screen of PROTERAN

4.4.2 Main Screen

If options 4.4.1.1 or 4.4.1.2 are selected, the main window with a view of the terrain with pattern 7 in front is displayed as shown in Figure 4-5.

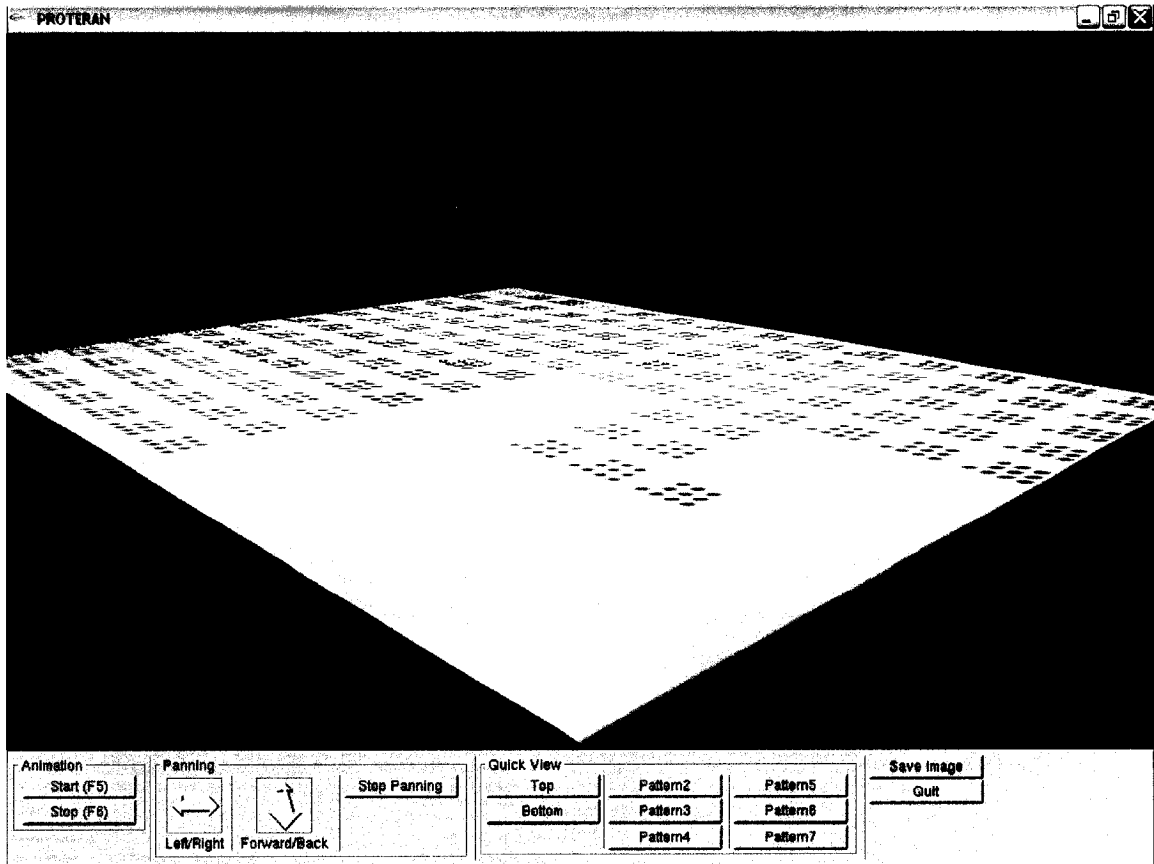


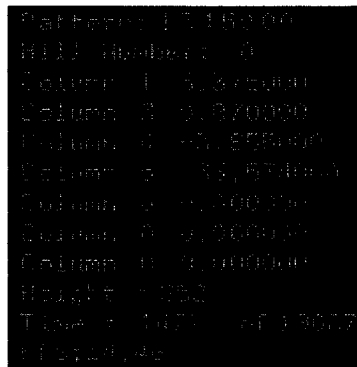
Figure 4-5: Main Screen of PROTERAN

4.4.2.1 Fly Through and Landing on Mountain:

Clicking the left mouse button and dragging while in the display, changes the user's viewpoint. By dragging the mouse left, right, up or down the view changes respectively. To move left, right, forwards or backwards the arrow keys are used. By clicking and dragging the left mouse button while simultaneously pressing the arrow keys the user can navigate "quickly". Also by clicking on a mountain the user lands on it and obtains the values of that patterned cluster.

4.4.2.2 Data Caption:

The top left corner displays information about the terrain and contains the information of the mountain clicked by the user (*cf.* Figure 4-6).



```
Pattern: 1315000
Hill Number: 0
Column 1: 5,370000
Column 2: 0,070000
Column 3: 0,055000
Column 4: 33,570000
Column 5: 0,000000
Column 6: 0,000000
Column 7: 0,000000
Column 8: 0,000000
Height: 000
Time: 1073 of 19017
13150,40
```

Figure 4-6: Data Caption

4.4.2.2.1 Pattern:

Represents the column combination, 1345 in Figure 4-5.

4.4.2.2.2 Hill Number:

Represents which of the top 10 occurrences of the column combination the user clicked, this value can vary between 0 and 9. A value of 0 indicates the center mountain or the highest occurring mountain was clicked, while a value of 9 means the tenth most occurring pattern was selected.

4.4.2.2.3 Column x:

The value of column x in the pattern.

4.4.2.2.4 Height:

Height of the mountain. Upon termination this equals the number of occurrences in the patterned cluster.

4.4.2.2.5 Time:

The length of animation. The left value indicates how many time units or experiments have been completed. The value on the right indicates the total time units or total number of experiments.

4.4.2.2.6 FPS:

Displays the average number of times the scene is rendered to display. Faster CPUs and optimized graphic cards have higher frame rates and result in a visually smoother display.

4.4.2.3 Buttons:

Many buttons have been implemented to increase usability of **PROTERAN** (cf. Figure 4-7). An explanation of these features follows.

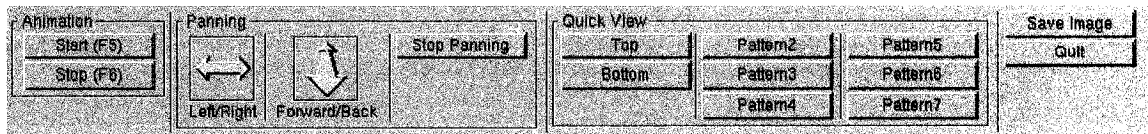


Figure 4-7: Buttons used in **PROTERAN**

4.4.2.3.1 Animation

4.4.2.3.1.1 Start:

Resumes or begins evolution of the mountains. When the final time unit is reached, the animation stops. A successive press of the 'Start' button will start the animation over.

4.4.2.3.1.2 Stop:

Pauses the animation. Animation is restarted with a press of the 'Start' button.

4.4.2.3.2 Panning:

4.4.2.3.2.1 Left/Right:

By dragging the mouse over this button the user can continuously turn left or right.

4.4.2.3.2.2 Forward/Back:

By dragging the mouse over this button the user can continuously move forward or backwards.

4.4.2.3.2.3 Stop Panning:

Stops from turning left/right or from moving forward/back.

4.4.2.3.3 Quick View:

Enable the user quick access to specific views of the terrain.

4.4.2.3.3.1 Top & Bottom Views:

Clicking on these buttons will give a view of the top or bottom of the terrain.

4.4.2.3.3.2 Pattern 2-7:

These buttons will give a view directed at the pattern type clicked.

4.4.2.3.4 Miscellaneous:

4.4.2.3.4.1 Save Image:

Clicking this button will save the current image in the frame buffer as a TGA file. The file will be saved as the name of the file inputted by the user with a number.tga attached to the end of it.

4.4.2.3.4.2 Quit:

Closes **PROTERAN**.

4.4.3 Other features:

The color-coding of the terrain enables the user quick identification of the different Pattern Types (*cf.* Table 4-5).

Table 4-4: List of the Pattern Type Colors

Pattern Type	Color
2	RED
3	GREEN
4	TURQUIOSE
5	BLUE
6	PURPLE
7	YELLOW

4.5 Testing and Release

PROTERAN was initially tested for small subsets of sample data provided by the user. Subsequently the entire data was loaded and when the program was successfully finished it was released for use by Dr. Parida and Dr. Zhou.

CHAPTER 5 - Conclusions and Extensions

5.1 Initial Experience with PROTERAN

At the time of writing of this thesis, **PROTERAN** has been in use for about two months. While the use has not been extensive and also primarily for visualizing the β -hairpin data set, discussions with the users has provided very encouraging feedback. Their experience with **PROTERAN** is summarized below:

By placing each of the patterns in a static place Dr. Parida and Dr. Zhou were able to easily identify the states of a protein. By landing (clicking) on the mountains they could easily obtain the values of the respective states. In addition because the hills grow over time they could easily see which states occur before other states. As quoted by Dr. Zhou *“This is extremely useful in identifying time sequences of the intermediate states in protein folding. For example, in this particular beta-hairpin case, we have noticed that some of the 2-column patterns (column 1, 4) occur before the 3-column patterns (column 0, 1, 4). The column 0 represents the native beta-strand H-bonds, while column 1 represents the radius gyration of the hydrophobic core and column 4 the fraction of native contacts. This indicates that the hydrophobic core is largely formed before the beta-strand hydrogen bonds are formed”*.

Another advantage of using the terrain metaphor was that by just looking at the heights of the mountain they were able to determine what the major states are. Furthermore zoom in and out buttons enabled them to get a global view of the data as well as the ability to focus on a specific patterned cluster. Finally the ability to “fly” through the terrain allowed them to get different perspectives on the large dataset.

The following remarks by Dr. Zhou indicate the future of **PROTERAN**. *“It should be pointed out that the current data are from Monte Carlo simulations (REM is essentially a Monte Carlo method even though it runs molecular dynamics in the underlying sampling), so the time sequence is not in real time, nevertheless, we are able to deduce some of folding process from these thermodynamic data. We expect **PROTERAN** will be even more useful in analyzing the real time data from straight molecular dynamics simulations using supercomputers, such as IBM’s Blue Gene machine.”*

5.2 Extensions

It can be seen that from the previous section that **PROTERAN** in its current version appears to be quite useful. There are many extensions that could make this system more generic and increase its potential for much wider use in the bio-informatics domain.

1. The first of these is to acquire different types of protein data and analyze their results using **PROTERAN**. This should provide us with the necessary feedback to improve the geometry mapping algorithm as well as make further improvements to the user interface.
2. Currently **PROTERAN** supports the use of up to and including 7 reaction coordinates. Future requirements may increase this number. This increase in the number of characteristics used will need a different layout of the terrain. The current layout design was based on a manual placing of the

pattern types. We need to develop a generic automated layout technique which future versions of **PROTERAN** should implement.

3. Finally it remains to be seen if **PROTERAN** can be used to visually analyze other types of data other than proteins. Because **PROTERAN** visualizes clusters of characteristics anything that can be defined by multiple characteristics over time can utilize **PROTERAN**.

References

- [1] Department of Energy. *Primer On Molecular Genetics*. Washington, DC: U.S. Department of Energy, Office of Energy Research and Office of Health and Environmental Research, 1992.
- [2] Karp, Gerald. *Cell and Molecular Biology: Concepts and Experiments*. 2nd ed. Toronto: John Wiley & Sons, Inc., 1999.
- [3] Wikipedia, the free encyclopedia. 2003. "Gene." [online] Available on the WWW at <<http://en.wikipedia.org/wiki/Gene>>.
- [4] Shing, Leming. 2002. "DNA Microarray (Genome Chip): Monitoring the Genome on a Chip." *www.Gene-Chips.com*. [online]. Available on the WWW at <<http://www.gene-chips.com/GeneChips.html#What>>
- [5] "Affordable three-color microarray scanner" [online]. Available on the WWW at <<http://www.laboratorytalk.com/news/gti/gti128.html>>
- [6] "DNA Microarray Methodology" [online]. Available on the WWW at <<http://www.bio.davidson.edu/courses/genomics/chip/chipQ.html>>
- [7] Sturn , Alexander. "Cluster Analysis for Large Scale Gene Expression Studies." Masters thesis. The Institute for Genomic Research, 2000.
- [8] "The Quest for the Protein Chip." *The Economist*. March 13, 2003. [online]. Available on the WWW at <http://www.economist.com/science/tq/displayStory.cfm?Story_id=1620807>
- [9] Shamir, Ron and Sharan, Roded. "Algorithmic Approaches to Clustering Gene Expression Data." In *Current Topics in Computational Biology*, edited by T. Jiang, T. Smith, Y. Xu, M.Q. Zhang. MIT Press, 2002.

- [10] “A tutorial on Clustering Algorithms.” [online]. Available on the WWW at <http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial.html>
- [11] Eisen, Michael B., Paul T. Spellman, Patrick O. Brown, and David Botstein. 1998. Cluster Analysis and display of genome-wide expression patterns. *PNAS USA* 95 (December): 14863-14868.
- [12] Schroeder, Michael and Katopodis, George. “Can hierarchical clustering improve the efficiency of non-linear dimension reduction with spring embedding?” In *Proceedings of the ECML/PKDD Workshop on Visual Data mining, Helsinki, Finland, 2002*, edited by S.J. Simoff, M. Noirhomme-Fraiture, and M.H. Böhlen. 2002.
- [13] Craven, Mark. 2002. “Clustering Gene Expression Data (Part 1).” *Biostatistics & Medical Informatics at UW-Madison*. [online]. Available on the WWW at <http://www.biostat.wisc.edu/bmi576/fall-2002/lecture15.pdf>
- [14] Tamayo, P., D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovky, E.S. Lander, and T.R. Golub. 1999. Interpreting patterns of gene expression with self-organizing maps: Methods and application to hematopoietic differentiation. *PNAS USA* 96 (March): 2907-2912.
- [15] Koren, Yehuda and Harel, David. “A two-way visualization method for clustered data.” In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington D.C.* ACM Press, 2003.
- [16] “Self-Organizing Maps.” *Helsinki University of Technology*. [online]. Available on the WWW at <http://www.cis.hut.fi/research/som-research/>
- [17] “Data Analysis and Clustering.” *Stanford Microarray Database Help*. [online]. Available on the WWW at <http://smd.stanford.edu/help/clustering.shtml>

- [18] Wu, Hank. 2004. "Cluster Analysis and Visualization." [online]. Available on the WWW at <<http://gap.stat.sinica.edu.tw/Talks/Hank-ClusterVisualization.pdf>>
- [19] Hautaniemi, S., "Analysis and Visualization of Gene Expression Microarray Data in Human Cancer using Self-Organizing Maps." *Machine Learning* 52 (2003): 45-66.
- [20] Kaski, S., Nikkilä, J., Törönen, P., Castrén, E., Wong, G. "Analysis and Visualization of Gene Expression Data using Self-Organizing Maps." In *Proceedings of NSIP-01, IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing*. 2001.
- [21] Cheng, Yizong and Church, George M. "Biclustering of Expression Data." In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, 93-103. AAAI Press, 2000.
- [22] Huang, Zhexue. "A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining." In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (SIGMOD-DMKD 1997), Tucson, Arizona, May 1997*.
- [23] Chueng, Y.M. "k*-Means: A new generalized k-means clustering algorithm." *Pattern Recognition Letters* 24 (2003): 2883-2893.
- [24] Schroeder, M., Gilbert, D., van Helden, J., and Noy, P. 2001. Approaches to visualization in bioinformatics: from dendrograms to Space Explorer. *Information Sciences: An International Journal* 139 (November): 19-57.
- [25] Gene Network Sciences, Inc. *BiCluster - Microarray Data Analysis: Direct Gene-Sample Correlations*. 2001.
- [26] Yang, J., Wang H., Wang, W., and Yu, P. "Enhanced Biclustering on Expression Data." In *Proceedings of the 3rd IEEE Conference on Bioinformatics and Bioengineering (BIBE)*, 321-327. 2003.

- [27] Parida, L., Zhou, R., and Feng, J. "Protein Folding Trajectory Analysis using Patterned Clusters." Under submission.
- [28] Zhou, Ruhong, Bruce J. Berne, and Robert Germain. 2001. The free energy landscape for β hairpin folding in explicit water. *PNAS USA* 98 (December): 14931-14936.
- [29] Kim, S.K., Lund, J., Kiraly, M., Duke, K., Jiang, M., Stuart, J.M., Eizinger, A., Wylie, B.N., and Davidson, G.S. 2001. A gene expression map for *Caenorhabditis elegans*. *Science* 293 (September): 2087-2092.
- [30] Chalmers, M. "Using a Landscape Metaphor to Represent a Corpus of Documents." In Proceedings of the European Conference on Spatial Information Theory 93, edited by Frank, A., and Campari, I., 377-390. 1993.
- [31] "OpenGL. The Industry's Foundation for High Performance Graphics" [online]. Available on the WWW at <www.OpenGL.org>
- [32] Grogono, Peter. *Getting Started with OpenGL*. Montreal: Concordia University, 2003.

Appendix

APPENDIX A

Below lists some of the most commonly used similarity distances [7]:

Let x and y be n -component vectors with \bar{x} and \bar{y} being the mean of each vector respectively.

1. Pearson correlation coefficient

$$\frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (\text{A.1})$$

2. Uncentered Pearson correlation coefficient

$$\frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (\text{A.2})$$

3. Squared Pearson correlation coefficient

$$\left(\frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \right)^2 \quad (\text{A.3})$$

4. Averaged dot product

$$\frac{1}{n} \sum_{i=1}^n x_i y_i \quad (\text{A.4})$$

5. Covariance

$$\sum_{i=1}^n \frac{(x_i - \bar{x})(y_i - \bar{y})}{(n-1)} \quad (\text{A.5})$$

6. Euclidian distance

$$\sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (\text{A.6})$$

7. Manhattan distance

$$\sum_{i=1}^n |x_i - y_i| \tag{A.7}$$

APPENDIX B

Given a set denoted as X

$$X = [1\ 2\ 4\ 6\ 12\ 15\ 25\ 45\ 68\ 67\ 65\ 98]$$

there are many statistics that could be obtained from this set, such as mean, standard deviation, variance and co-variance. On the set X the mean would be defined as X bar and would be denoted with the following formula.

$$\bar{X} = \frac{\sum_{i=1}^n X_i}{n} \quad (\text{B.1})$$

The mean is a middle point but from the following two examples it does not reveal how much the data varies. For example both of the datasets below have a mean of 10 but dataset 1 varies a lot more than dataset 2.

$$[0\ 8\ 12\ 20] \text{ and } [8\ 9\ 11\ 12]$$

Standard deviation offers a solution to this problem and is calculated using the following formula:

$$s = \sqrt{\frac{\sum_{i=1}^n (X_i - \bar{X})^2}{(n-1)}} \quad (\text{B.2})$$

Dataset 1 has a standard deviation of 8.3266 while Dataset 2 has a standard deviation of 1.8257. Variance is also another term that is essentially the same as standard deviation.

$$s^2 = \frac{\sum_{i=1}^n (X_i - \bar{X})^2}{(n-1)} \quad (\text{B.3})$$

Standard deviation and variance are only applicable to one dimension whereas covariance is always applied to two dimensions. This is an important property for analyzing multivariate data. See (A.5) for formula of covariance:

Note that covariance can be applied on the same set $\text{cov}(X,X)$, this would simplify to the variance of X . Basically covariance is a statistic representing the degree to which two variables vary together.

For an n -dimensional data set $\frac{n!}{(n-2)! * 2}$ different covariances can be calculated.

All these measurements can be represented by an $n \times n$ matrix called a covariance matrix denoted as.

$$C^{n \times n} = (c_{i,j}, c_{i,j} = \text{cov}(\text{Dim}_i, \text{Dim}_j)) \quad (\text{B.4})$$

Two other mathematical properties Eigenvectors and Eigenvalues are involved in PCA. Eigenvectors are vectors that when multiplied by a square matrix produce a multiple of the original vector. It must be noted that not all square matrices have eigenvectors and if they do then an $n \times n$ matrix will have n eigenvectors. Each eigenvector produces a multiple of the original vector; this multiple is known as the eigenvalue and always produces the same result no matter how much the eigenvector is scaled. Another important property of eigenvectors is that they are orthogonal to each other.