2-D AND 3-D LAYOUTS TO AID HUMAN COGNITION OF LOCAL STRUCTURE IN MULTIVARIATE DATA

by

Ru Sun

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computational Sciences and Informatics

Committee:

Dr. Daniel Carr, Dissertation Director

Dr. James Gentle, Committee Member

Dr. Igor Griva, Committee Member

Dr. James Willett, Committee Member

Dr. D. Papaconstantopoulos,
Department Chairperson

Dr. Peter Becker, Associate Dean
for Graduate Programs, College of Science

Dr. Vikas Chandhoke, Dean,
College of Science

Date: August 1, 2008

Summer Semester 2008
George Mason University
Fairfax, VA

2-D and 3-D Layouts to Aid Human Cognition of Local Structure in Multivariate Data

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Ru Sun
Master of Science
George Mason University, 2000

Director: Daniel B. Carr, Professor
Department of Statistics

Summer Semester 2008
George Mason University
Fairfax, VA

DEDICATION

This is dedicated to my God, my loving parents and my friends.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor Professor Daniel B. Carr for his supervision and constant guidance, support, encouragement, trust, and great patience throughout the years. He is a mentor and a friend. I was so fortunate to work with him.

I wish to thank my group leaders Tom Neubig and Mary Batcher at Ernst & Young, LLP, who granted me a flexible working arrangement, which made it possible for me to dedicate more time and energy in research. My colleagues were very supportive and covered many tasks for me at work. I am very grateful.

I am also fortunate to have a loving family and a large group of friends who pray for me, care about me and support me in every possible way. I can never thank them enough.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

2-D AND 3-D LAYOUTS TO AID HUMAN COGNITION OF LOCAL STRUCTURE IN MULTIVARIATE DATA

Ru Sun, Ph.D.

George Mason University, 2008

Dissertation Director: Dr. Daniel B. Carr

This dissertation addresses the development of new 2-D and 3-D layout algorithms for statistical visualization purposes. These layouts serve tasks that include placing near neighbors close together, showing group or cluster membership, allocating space for glyphs and images used to characterize objects (cases), and approximating distances between objects. These tasks serve goals that include conveying structure, facilitating pattern discovery and hypothesis generation, and providing access to detailed information. The layouts are for human use, so they include considerations of human perception, cognition, and organizational regularity.

This dissertation targets applications involving the study of cases, variables, clusters, and other multivariate objects. In these applications the notion of distances/dissimilarities between objects is important. However, accurate distances can not be maintained in low dimensional views. Researchers have developed a variety of

layout methods to represent multivariate objects (including data summaries) in low dimensions. Common layout algorithms include multidimensional scaling, Kohonen self-organizing maps, Treemaps and spring models. This dissertation compares and contrasts the new layout algorithms with previous methods, develops new star glyphs, and demonstrates the new algorithms using multivariate data produced by AIRS (Atmospheric InfraRed Sounder) and other datasets.

Chapter 1. Introduction

1.1 Problem Statement

The amount of available digital data in the world is growing exponentially. The sources of data include monitoring studies of human behavior and public opinions, survey studies of the environment, satellite datasets in earth science, gene expression datasets in bioinformatics and simulation studies of virtually anything imaginable. At the same time, our ability to extract understandable summaries and insights from such studies lags far behind. While data mining and automated modeling summarizations are progressively employed, there is an ever-increasing role for visualization to address data curation, pattern discovery, model construction, model criticism and communication. Visualization techniques are becoming increasingly important for the analysis and exploration of the deluge of large multidimensional data sets. The pioneering National Science Foundation (NSF) report by McCormick et al. (1987) stated: “Visualization is a method of computing.” Here the visualization refers to the process of transforming information into a visual form, enabling researchers to perceive and comprehend the content and structure of information. This emphasizes the importance of human visual perception, cognition, understanding and decision making. Clearly, we need to continually refine computational methods to serve these purposes.

Card et al. (1999) indicated, “Used effectively, visualization can accelerate perception of data. By designing visualizations with human strengths and weaknesses in mind, it is possible to exploit people's natural ability to recognize structure and patterns, and circumvent human limitations in memory and attention.” Chen (2006) declared, “The holy grail of information visualization is to make the insights stand out from otherwise chaotic and noisy data.” There are intimate connections among seeing, thinking and understanding.

Human sensation, perception, cognition and understanding have limits. For example, visible light is only a narrow band in the electromagnetic spectrum. Mankind has invested tremendous efforts to develop sensors for the other parts of the spectrum and to produce transformations that convert the results into the visible spectrum. However, research in converting patterns in data into cognitively useful forms has lagged behind. We humans live in a 3-D world and see surfaces with shape, color, and texture through mediums such as air or water. Even though we can imagine a world with higher dimensions and can think with multivariate models, the external input to our eyes is limited to moving surfaces in the visible spectrum. In exploratory data analysis, most plots are shown on a 2-D monitor using colored pixels in order to convey attribute relationships in space and time. Analysts view the plots, draw conclusions, and decide what to do next. The goal is to encode the data variables so that the important patterns, structures, statistics or models can be noticed and comprehended by the carbon-based computer that is the human mind.

While it is not possible to fully comprehend high dimensional structure in low dimensional layouts, layouts can help reveal some structures and occasionally all of the structures when they are very simple. Low dimensional structure is often embedded in higher dimensions. Furnas and Buja (1994) discuss how compositions of projections (display aspects of structure in low dimensions) and sections (constraints in high dimensions) can display aspects of structure of modest dimension.

A paper written by Ankerst, Berchtold and Keim (1998) discussed the spatial dimension arrangement problem. They considered the problem of finding an optimal one or two-dimensional arrangement of the objects on a regular grid based on their similarities. Theoretical considerations show that both the one and two-dimensional arrangement problems are often computationally hard problems, i.e., they are NP-complete. Usually heuristic algorithms are used. This keeps the door open to the continuous development of heuristic algorithms intended to address specific layout tasks.

Layouts are often regular grids, lattices or quasi-regular structures that approximately preserve properties such as cell roundness. The challenge here is to develop/refine regular layouts and glyphs to show many objects (cases) or clusters and a modest number of variables to help analysts see meaningful patterns that might otherwise be missed.

The regularity of layouts can make the graphics seem more accessible. Tufte (1983) describes small multiple layouts. He says that the advantage of small multiples is that once viewers understand the design of one piece, they can immediately access data in all other pieces. The constancy of the design allows viewers to focus on changes in the

data rather than changes in the design, thus speeding up the information conveyance. The regularity of lattice points and grid cells is also advantageous.

Card et al. (1999) discuss 2-D versus 3-D spatial encodings. It is clear that more cases can be shown in 3-D than 2-D. Nonetheless, visualization in 2-D is commonly accepted as the effective way for presenting data. Compared to 2-D, 3-D graphs suffer from the fundamental problem of occlusion, where objects closer to the viewer can hide distant objects, and dense collections of objects can obscure their distributions. Transparency (via alpha blending) and masking provide limited help. 3-D graphs also involve more challenges both in implementation and design. 3-D implementations require significantly more processing power and involve more parameters to define objects, surface properties, lighting models, locations, and motions. However, advances in computing power and rendering hardware have substantially addressed the computing problem.

2-D graphs are often displayed on a computer monitor and continuing problem is the limited screen resolution, even though monitors in the market keep improving their resolutions. Often there are not enough pixels on the monitor to display all the information. A basic need is more visual space. Printed hard copies do a lot better than a screen due to their much higher resolution. A serious option to show more cases is to make poster-size plots. Our visual and cognitive processes emphasize a small part of our field of view. A large plot lets us shift our focal point to quickly refresh our limited visual memory. Two obvious weaknesses of large printed plots are their production time and lack of re-expression options offered by interactive and dynamic graphics. Other

weaknesses are that comparison accuracy decreases with distance, and that large plots can appear intimidating.

Glyphs are graphical objects whose elements (e.g., position, size, shape, color, orientation, etc.) are bound to multivariate values that describe details of the cases. Star plots and Chernoff faces are among the more popular glyphs, and even time series lines can be considered as glyphs. Historically much glyph plotting used a row and column layout with a happenstance ordering. With such layouts, it is next to impossible to make simple observations concerning the presence of clusters or to note if points fall close to low dimensional manifolds. It seems that better organization of glyphs is attainable, but criteria need to be chosen and algorithms to be developed to produce the layouts.

When the glyphs are round, the row and column layout is less than ideal. Such plots invoke the metaphor of putting round pegs in square holes. Carr (1991) discusses the geometric and visual merits of hexagon and truncated octahedron grids in 2-D and 3-D. They are the roundest regular polygons and polytopes that tessellate the respective dimensions. The number of neighbors, 6 and 14, is based on the number of shared faces and is unambiguous. With squares and cubes the number of neighbors is only 4 and 6 when faces are considered. Considering polygons or polytopes with shared vertices as neighbors is less natural. This research addresses the challenge of developing layouts better suited for round glyphs.

Another important layout task is to help analysts comprehend and assess the results of clustering algorithms. For modest data sets, analysts often use a tree-like plot called a dendrogram to show the results of hierarchical clustering (Everitt 1993, Johnson

and Shneiderman 1991, Gnanadesikan, Blashfield et al. 1989). The typical dendrogram provides one-dimensional rank order for the cases. The default rank order (seriation) produced by many algorithms reflects both the joining of groups and the original order of the cases. This is not optimal since groups that are potentially far apart are forced to be adjacent. While in most cases two-dimensional and three-dimensional layouts do not fully solve the problem, the additional dimensions provide more flexibility for placing similar points near each other. The problem is well-known and has been addressed using several approaches.

The notion of permuting the rows and columns of a matrix is not new. Bertin (1984) has an example diagram that illustrates how the diagonalization process of a table of correspondences between two lists helps to reveal structure. Carr and Olsen (1996) discuss the merit of sorting rows and columns in two-way layouts of cases and variables. They suggest using ordering methods other than those based on dendrograms. Their preferred method is a height directed preorder traversal of a minimal spanning tree. When variables are comparable units or transformed to be comparable units, Cleveland (1993) suggests ordering cases based on the median of their multivariate values. The two-way ordering problem can also be addressed using singular value decomposition (SVD). The first left eigenvector can be used to order cases, and the first right eigenvector can be used to order variables. Two-way ordering leads to the correspondence analysis, and its solution can also be found using SVD, as shown in Greenacre (1984).

Graphical layout issues need to be considered when designing plots. There are cognitive and design principles to follow. Perceptual considerations are necessary to

design a clear and attractive layout. The goal is to exploit human perception and cognitive processing capabilities and avoid weaknesses. For example, clusters are separate, so I convey this in the cluster layouts by using white space (gaps) to show the separations. The gap widths can also weakly encode the distances among adjacent clusters. It helps researchers perceive the number of clusters in the data at first glance. 3-D cues can also enhance the perception of gaps between clusters.

1.2 Contributions

This dissertation enhances Wills' (1998) recursive partitioning algorithm for clusters by adding gaps to visually separate clusters at each level of partitioning. It also develops three new 2-D layout algorithms that allocate at most one case per cell.

1) The Compact Area Partitioning Algorithm (CAP) recursively partitions a convex polygon to produce round regions to contain the cases;

2) The Hexagon Occupant Exchange Algorithm (HOE) exchanges cases in hexagon cells to reduce a cost based on the distances of each case to cases in the neighboring cells. The algorithm supports different cost functions;

3) The Hexagon Cluster Layout Algorithm (HCL) uses a multidimensional scaling start followed by a spring model to find a centroid location for each cluster in a hexagon grid. The algorithm then lays out cases one per cell around the centroids and rearranges the cases within each cluster in a manner similar to item 2). Some cells are left empty; and

4) This dissertation extends the CAP algorithm to 3-D, where it recursively partitions a convex polytope to produce small round polytopes where a glyph for each case can be plotted.

Another contribution involves designing two special star glyphs, namely signed deviation star glyphs and folded deviation star glyphs. They provide a more symmetric encoding of large and small values than traditional star glyphs, and show variables with increased perceptual accuracy of extraction. I indicate that graphics produced using these methodologies have merits based on cognitive principles and graphical design considerations when compared to the alternatives. The layouts are illustrated in new applications.

The Compact Area Partitioning Algorithm (CAP) is a new layout approach for visualizing cases that have been clustered. It recursively divides a convex polygon to layout clusters, so that the areas of the sub-polygons are proportional to the sizes of the sub-clusters, and the sub-polygons are as round as possible. The goal is to visually emphasize the clusters and produce round regions to contain glyphs or links that provide access to case details. Distinctive features of this algorithm include 1) use of area partitioning lines with more orientations than the horizontal and vertical lines used in the traditional Treemap algorithm, 2) selection of splits that create two polygons with the smallest sum of dimensionless second moments about their centroids, and 3) optional use of white space gaps to indicate the distance between the two groups to be laid out in the two polygons created. The approach addresses the problem of slivers that can result when

horizontal or vertical partitioning of a rectangle to produce two areas of correct proportions making one side of one rectangle smaller than a pixel.

Both the Hexagon Occupant Exchange Algorithm (HOE) and the Hexagon Cluster Layout Algorithm (HCL) are developed to layout multivariate objects (cases) on a hexagon lattice. In HOE, cases are initially assigned randomly to hexagon lattice cells so that some of the cells are occupied by cases and the remaining cells are empty. The algorithm randomly selects pairs of cells. The occupants of the two selected cells are exchanged if the change reduces the overall cost. The cost is a function of the dissimilarities among the occupants of the neighboring cells. When both of the neighboring cells are occupied by cases, the dissimilarity between the cases is used. Pre-set dissimilarity values are used when at least one of the neighboring cells is empty. In HCL, a spring model applied to a multidimensional scaling start determines the centroid location for each cluster, The locations are mapped to a hexagon lattice. The algorithm then lays out the cases or case representations in cells of expanding concentric circles about the centroids. The spring model is set so that white space separates the clusters. The cases within each cluster are rearranged so that similar cases are close together.

I extend the CAP algorithm to 3-D to recursively partition a convex polytope. The exact geometrical equations and calculations are complicated, so I generate points on a body-centered cube lattice and perform approximate empirical calculations to partition the polytope of points. 3-D layouts often preserve local distances better than 2-D, although they might be harder to visualize. The 3-D graphics tools used in this research include Glisten and RGL and will be discussed in the context.

This dissertation also reviews various modern clustering methods as well as layout algorithms, such as multidimensional scaling (Borg 1997, Kruskal and Wish 1978), self-organizing maps (Kohonen 1995, Hulle 2000), spring models (Eades 1984), seriation methods (Marcotorchino 1991), treemaps (Shneiderman, 1992) and extensions, etc. Various similarity measures, cluster layout issues, computational and graphical presentation issues are also discussed.

This dissertation develops practical graphical displays that are helpful in the exploratory analysis of multivariate data sets and are suitable for presentation and communication settings. The emphasis is on the representations that show objects with a locally unified encoding such as a star glyph. This contrasts with linked representations, such as scatterplot matrices, linked micromaps and parallel coordinates plots that convey facets of the same object at different locations on the screen or other plotting devices.

The methodology developed here applies to situations involving cases for which dissimilarities or distances are available. This covers the common n cases by p variables data table where the variables are used to assess dissimilarity or distance between cases. However, each case can be a summary or organizational representation of objects. For example, a case could be a dendrogram which is an organizational representation for objects that might in other contexts also be called cases. The cases can also be data subsets, clusters, trees, models, processes or anything else, as long as meaningful similarity/dissimilarity measures between cases can be assessed. Thus the algorithms are general and can be applied in various settings. An extreme example starts with many terabytes of satellite data. The resulting geophysical parameters over space and time are

aggregated within 5 degree latitude and longitude grid cells of the earth and summarized as multiple multivariate multi-altitude clusters. These multiple summary clusters for each grid cell provide the basis for computing the expected distance between cells. This is used to cluster earth grid cells that are then organized by layout algorithms.

1.3 Organization

In this chapter, the problem and goals have been presented. The rest of the dissertation is organized as follows. In Chapter 2, background and related methods are introduced. In Chapter 3, recursive partitioning 2-D layouts and the CAP algorithm are discussed. Chapter 4 covers the 2-D HOE and HCL algorithms. The extension of the CAP algorithm to 3-D is presented in Chapter 5. Chapter 6 provides conclusions and indicates future work.

Chapter 2. Background

Several layout algorithms are used to represent multivariate information in 2-D or 3-D plots. The better known algorithms include multidimensional scaling, spring models, self-organizing maps, treemaps, dendrograms, and heat maps. Some layouts are designed for specific purposes. This chapter provides an overview and brief discussion of selected layout algorithms and related issues that motivate the new algorithms in chapters 3 and 4.

2.1 Singular Value Decomposition and Multidimensional Scaling

2.1.1 Singular Value Decomposition (SVD)

SVD is a standard decomposition in numerical analysis. An $n \times p$ matrix X has a SVD of the form $X = UDV^T$, where U and V are matrices of orthonormal columns, and D is a diagonal matrix. Conventionally the diagonal elements of D (called singular values) are sorted in decreasing order. In this case, D is uniquely determined by X. The number of non-zero diagonal elements is the rank of X.

SVD can be used in calculating matrix pseudoinverses, matrix approximations, in solving linear equations and in least square problems, etc. It finds applications in many

fields, such as digital signal processing, image processing and bioinformatics. SVD is the foundation of popular statistical methods for dimensionality reduction, including principal component analysis, factor analysis, correspondence analysis and multidimensional scaling. It is also used in latent semantic indexing for documents analysis. The dimensionality reduction techniques are highly relevant in the task of data visualization, since the goal is to show multidimensional objects in lower dimensional space.

2.1.2 Multidimensional Scaling (MDS)

Multidimensional Scaling (MDS) is a set of related mathematical techniques that enable researchers to reveal the ‘hidden structure’ of data sets. MDS has its origins in psychometrics, where it was proposed to help understand people's judgments of the similarity of members of a set of objects. Each object is to be represented by a data point in a multidimensional space. The goal of MDS analysis is to represent assessments of dissimilarity among pairs of objects as distances between points in a relatively low dimensional geometrical space, so that the distances between pairs of points have the strongest possible relationship to the corresponding dissimilarities among the pairs of objects. Ideally, two similar objects are represented by two points that are close together in the new space, and two dissimilar objects are represented by two points that are far apart. Given a dissimilarity matrix with rank *p*, metric MDS can produce a set of *p*-variate points whose Euclidean distance matrix matches the dissimilarity matrix. Typically MDS produces lower dimensional points whose Euclidean distance matrix is

an approximation to the dissimilarity matrix based on the Frobenius norm. The visualization space is usually a two or three-dimensional Euclidean space, but sometimes it may be non-Euclidean and may have more dimensions. The graphical display of the similarities among objects provided by MDS enables researchers to explore structure in low dimensional views.

MDS is a generic term that includes many different specific types and approximations. Non-metric scaling attempts to preserve the rank order of values in the dissimilarity matrix, while metric scaling attempts to preserve the dissimilarity matrix. The number of similarity matrices and the nature of the MDS model can also classify MDS types. This classification yields classical MDS (one matrix, unweighted model), replicated MDS (several matrices, unweighted model), and weighted MDS (several matrices, weighted model). Young (1985) has a more detailed discussion.

The metric MDS originated with Richardson (1938) and was developed and explicated by Torgerson (1958). Kruskal and Wish (1978) has a readable and brief introduction to MDS. Borg (1997) gives a comprehensive presentation. MDS has become a general data analysis technique and found its applications in scientific visualization and data mining in diverse fields such as cognitive science, information science, marketing, sociology, political science, psychology and biology.

With MDS, objects can overplot. Overplotting is a problem in visualization that should be avoided or kept under control. If the start is a MDS layout of cases, spring models can be applied to push cases apart, so they are capable of being seen as separated. Other algorithms such as generalization of Carr's (1994a) 1-D nudging algorithms could

also be used. Another overplotting avoidance approach puts each case in a different grid cell and then exchange cases. The use of MDS in this dissertation is to help produce a good start to layout cases in individual cells, so cases of each cluster are contiguous and clusters are separated.

2.1.3 Similarity/Dissimilarity Measures for Complex Structures

Often the needs arise to layout general objects as well as individual data points. The first step is to have an appropriate way of defining similarity (or dissimilarity) between the objects (cases). Obviously the layout will only be as meaningful as the input similarities.

There are many ways of measuring similarity/dissimilarity. Gentle (2005) has a detailed discussion on measures for numeric data, categorical data, and groups of observations. Often the term ‘distance’ is used loosely and interchangeably with ‘dissimilarity’, even though dissimilarity measures are not required to obey the triangle inequality, as distance measures do. Distances can be dissimilarities. There are several ways of measuring distance. Some common ones are Euclidean distance, maximum difference, Manhattan distance, binary difference, etc. Hammer, Harper and Ryan (2004) give 12 indices that can be used to compute the distance matrix. Data are usually scaled before calculating distances. Measures of similarity include covariance, correlation, rank correlation, and cosine of the angles between two vectors. Any measure of dissimilarity can be transformed into a measure of similarity by applying a monotonic decreasing function.

For trees, many people have proposed similarity measures based on the number of 'changes' that it takes to transform one tree into another. A 'change' is defined as an addition or subtraction to the branches or splitting variables in the tree.

Musser (1999) proposed two different measures for comparing trees, both of which are based on how well the data are fit by the tree:

- Correlation between fitted values
- Optimal 'node-matching'

The first measure is simply to compute the correlation coefficient between the fitted values of two trees. If the correlation is exactly 1, then the two trees are perfectly correlated. They produce the same fitted values and the partition of the data is the same, but their dendrograms might be different. A dendrogram is a tree diagram used to illustrate the arrangement of clusters.

The second measure is based on matching nodes between trees. Each observation is located in a single terminal node in each tree. Node-matching tabulates the number of observations in both terminal node *i* of tree *A* and in terminal node *j* of tree *B*. If the two trees were identical functionally, then this matrix of counts would be a permutation of a diagonal matrix.

These two measures can give very different pictures of the similarity between two dendrograms. It is possible that two dendrograms have mostly similar nodes but having different fitted values. On the other hand, it is possible for two trees to produce similar fitted values but the trees could group observations into very different nodes.

These two measures are not actual distance measures on the structure of the trees because they are not metrics. A proper distance measure requires that if the distance between two objects is 0, then the two objects must be the same. However, two dendrograms may have a correlation of 1.0 or they may have perfect node matching, and still not be identical trees.

When the data set is large, it may not be practical to display all of the original individual observations. A common approach first clusters the data, then treats the clusters as cases and finally lays them out in the same way that individual points are laid out. For this layout purpose, the distance between two clusters needs to be defined. Three classical ways of defining distance between clusters includes:

- the minimum distance between a point in one cluster and a point in the other cluster,
- the average of the distances between the points in one cluster and the points in the other cluster, and
- the maximum distance between a point in one cluster and a point in the other cluster.

The hierarchical clustering algorithms using the above three cluster distance definitions are called single linkage, average linkage and complete linkage methods respectively. The single linkage method tends to produce long, stringy clusters, while the complete linkage method is likely to form more spherical clusters. As a compromise, the average linkage method is the most widely used (Gnanadesikan and Blashfield et al. 1989).

Models or other multivariate objects may motivate the development of new measures of similarity or dissimilarity.

2.2 Self-Organizing Maps

A self-organizing map (SOM, Kohonen's map) is a type of artificial neural network that is trained using unsupervised learning to produce low dimensional representation of the training samples while preserving the topological properties. It was first proposed by Teuvo Kohonen in 1981 as a visualization tool, and has since been successfully used for the analysis and organization of large data sets in various fields (Kohonen 1995). The biological analogue is simple: There exists geographical correspondence of the visual and somatosensory cortices to the respective receptor surfaces. A topographical projection of the basilar membrane of the inner ear onto the auditory cortex might exist, but there are many other neural maps in the brain for which no ordered receptor surface exists. The only possibility remaining is that such abstract feature maps are formed in a self-organizing process, where the global order in a brain area emerges from local interactions around the active units during learning. Since an active cell tends to make its neighbor cells act in a similar way, a continuous map of different responses will be formed. During self-organization, neighboring neurons in the map cooperate and code for similar events in the input space, whereas more distant neurons compete and code for dissimilar events. The objective of the SOM algorithm is the organization of the input patterns into a topological structure represented by its neurons, where the relationships between different patterns are preserved (Hulle 2000).

With SOM, high dimensional datasets are projected onto a 1-D, 2-D or 3-D space. Typically, a SOM has a two or three dimensional lattice of neurons and each neuron represents a cluster. During the learning process, all neurons compete for each input pattern; the neuron that is the most similar to the input pattern wins and only the winning neuron is activated (winner-takes-all). The winner updates itself and the neighbor neurons to approximate the distribution of the patterns in the input dataset. After the learning process is complete, similar clusters will be close to each other. Topological ordering helps in detecting both distinct and similar clusters quickly. The SOM algorithm is efficient in handling large datasets. It has been used in data mining and exploratory data analysis applied to large databases of financial data, medical data, gene expression data, free-form text documents, digital images, human voice analysis, process measurements, etc. (Toronen 1999, Vesanto 2002).

The SOM cells are typically square or hexagon cells in a grid and will not be empty. The SOM toolbox developed for MATLAB can show the clustering result as a ‘U-matrix’. The U-matrix is constructed from the values of the neurons. Both the distance between neighboring neurons and the distance of a neuron (defined as the average of the distances between this neuron and all of its neighbors) are displayed in a U-matrix.

SOM can be viewed being similar to K-means clustering where K is specified by the number of cells in the layout. Clustering is a two-phase process: determining the number of clusters and clustering the data. Determining the number of clusters is not trivial since the characteristics of the data set are usually not known a priori. This can be accomplished by running the algorithm with varying numbers of clusters and selecting

the most appropriate clustering result. Trosset (2005) discusses and compares MDS, SOM, and K-means clustering.

The algorithm in effect combines clustering with cluster layout and makes the result hard to interpret in terms of what is known about the separate clustering and layout tasks. This makes it harder to think about the cluster structure in the data.

Hastie et al. (2003) indicate that in SOM, points close together in the original feature space should map close together on the low dimension manifold, but points far apart originally might also map close together. Guo et al. (2002) indicated that ‘subspace clustering methods’ such as multidimensional scaling or SOM have two major drawbacks: 1) new dimensions (as linear combinations of the original dimensions) and result clusters are hard to interpret; 2) they can not preserve clusters existing in different subspaces.

2.3 Space Filling Layouts - TreeMaps

Researchers have developed many methods for the display of hierarchical information structures, or for short, trees (cone tree by Robertson et al. 1991, botanical trees by Kleiberg et al. 2001, treemaps by Johnson and Shneiderman 1991). Trees arise in a recursive partitioning and agglomerative clustering context. There are two common conventions to present trees: a straight-line convention and a containment convention. The classic planar straight-line convention represents nodes as dots and edges as straight lines connecting the nodes. The containment convention represents nodes as closed regions with children nodes contained inside parent nodes.

The common approach for laying out the result of hierarchical clustering is a binary tree or dendrogram. A dendrogram is a tree diagram used to illustrate the arrangement of clusters, typically looks like this:

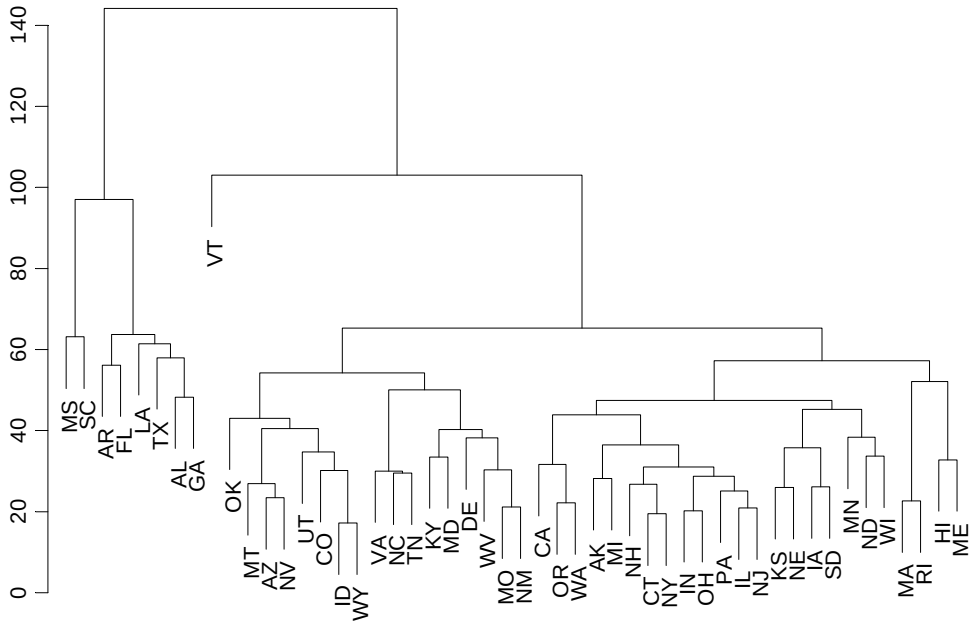

Figure 1. A Dendrogram Example

The dendrogram follows the first convention of drawing trees. The common dendrogram plots cases along one axis and use a second dimension to represent the hierarchical joining of cases into clusters and the joining of clusters with clusters. Some variants also use the second dimension to show the distance between clusters that are joined. There are many options for determining exactly how the cases are put in a linear

order while avoiding the crossing of lines in the dendrogram. The dendrogram defaults that are often used to separately order the rows and columns of microarray heat maps are subject to improvement.

Dendrograms and other node-and-link diagrams from the first convention have several problems. The linear ordering of cases limits the number of cases that can be shown in a display. They do not use the available display area effectively: their practical applicability is limited to small trees with at most a few hundred nodes, when the cut level changes the tree structure often changes dramatically, cluster size is not obvious, and the points appearing close in the dendrogram are not necessarily close in space.

While there are approaches that can deal with a large number of cases by showing some level of clustering and zooming in on selected clusters, the interest here focuses on the two-dimensional layouts. The use of two-dimensional layouts for points representing cases allows more cases to be shown without recourse to summarization in groups. The containment convention of drawing trees can increase the efficiency of space filling and reduce the visual complexity.

Treemaps developed by Johnson and Shneiderman (1991) are a space-filling method of visualizing large hierarchical datasets. It fills recursively divided rectangles with components of a hierarchy. A typical Treemap looks like this:

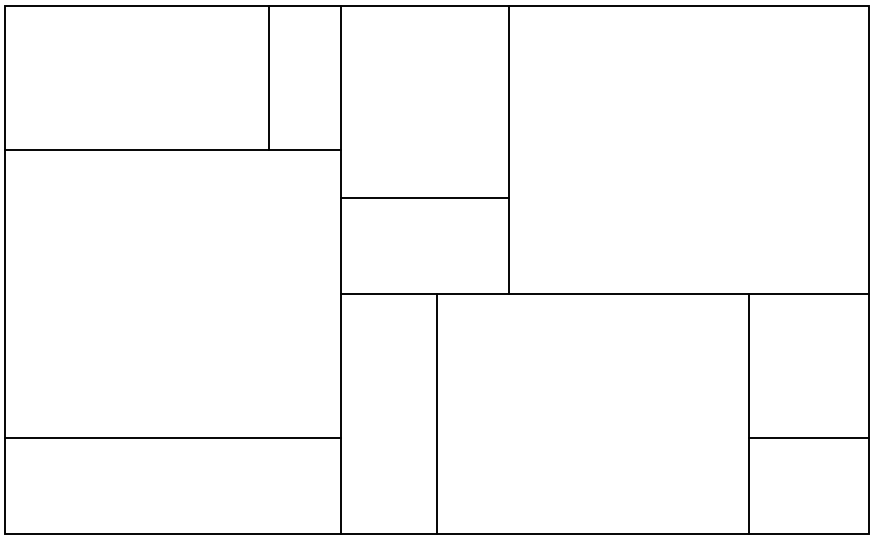

Figure 2. A Treemap Example

Treemaps follow the containment convention of drawing trees. A treemap works by creating a nested sequence of rectangles that make up the map. The Slice-and-Dice algorithm of the original treemap paper (Johnson and Shneiderman 1991) uses parallel lines to divide each parent rectangle into smaller children rectangles, with the areas of the sub-rectangles proportional to the number of nodes in the sub-clusters. The orientation of the partitioning lines (vertical or horizontal) is an option. Squarified treemaps choose the orientation to be parallel to the shorter sides of the parent rectangle. If the tree is not well balanced, as often the case when the data has outliers, the proportions can be so extreme that partitioning creates rectangles with a high aspect ratio called ‘slivers’. Slivers-like rectangles with a few pixels’ side or even less than a screen pixel side can be problematic in terms of color perception, the number of pixels of line separation relative to the

rectangles, the visualization of the rectangle centroid dots when centroids are displayed, and the lack of space to show glyphs or text. While a geospatial content is not the closest of analogies, the context helps reinforce the disadvantage of slivers. If we characterize the land surface of the continental US at one kilometer^2 resolution grid, the natural grid system would use 1 kilometer x 1 kilometer grid rather than a grid made up of rectangles of 1 meter by 1000 kilometers. Conway and Sloane (1999) indicate the merit of round regions in coding theory context. It is desirable to avoid generating slivers in the recursive partitioning of areas into fixed proportions. I and others (Sun and Carr 2005, Balzer and Deussen 2005, Wattenberg 2005b) have proposed several alternative layout algorithms to address this concern by reducing the overall aspect ratios.

The further developments of Treemap layout include Nested Treemaps (Johnson and Shneiderman 1991), Treemaps for Interactive View (Wills 1998), Clustered Treemaps (Wattenberg 1999), Cushion Treemaps (van Wijk and van de Wetering 1999), Squarified Treemaps (Bruls et al. 2000), Modifiable Treemaps (Vernier and Nigay 2000), Ordered Treemaps (Shneiderman and Wattenberg 2001, Bederson et al. 2001), etc. These variations improve different aspects of the original Treemaps, but are still restricted to a rectangular display area and the two axis-aligned partitioning directions.

As Wills (1998) indicates, the points representing cases in a two-dimensional space can be used in linked brushing (Becker and Cleveland 1987, Chen 2004, Martin 1995, Bartram and Ware 2002) to other graph representations for more detailed study. The other representations can include parallel coordinate plots, scatterplot matrices,

maps, and so on. Linked brushing is a very important tool for exploring multivariate data and identifying patterns and relationships among variables.

Another way to encode a few variables is to plot glyphs such as stars at the centroid of the cells in the space filling layout. If the cells in the cluster layout have too little area or are too thin, there is inadequate space for plotting glyphs. The alternative algorithms developed here that produce round and compact cells have merit in the glyph plotting context.

Wills (1998) describes a 2-D point layout algorithm to visualize the results of a general hierarchical clustering algorithm. The basic idea is to represent the cases belonging to a cluster as points inside a rectangle. His layout of case-representing points allows the rectangles to be drawn for any level of the hierarchical clustering without changing the position of points. The algorithm method starts with a specified rectangular area and recursively divides each rectangle into two smaller rectangles. The relative areas of the two rectangles are chosen to be proportional to the number of cases in the two sub-clusters being represented. Hierarchical clustering sequentially joins two clusters (including single case clusters) into one cluster so the process can be represented as a binary tree. Reversing the process, the algorithm starts at the root of the tree which represents a single cluster with all cases, determines the number of cases associated with the two sub-trees and repeats the process for subsequent sub-trees. When a rectangle contains just one case from a leaf node of the tree, the case is plotted at the center of the rectangle and there is no further partitioning of the rectangle. The algorithm partitions the

longer side of the rectangle in the effort to produce rectangles that are closer to squares. Nonetheless the algorithm can produce slivers.

Consider the case of starting with a 1280 x 1024 rectangle corresponding to pixels on a monitor and consider a layout involving 6000 cases with one extreme outlier that is joined with the rest of the cases at the very last step of the hierarchical agglomeration. Splitting the long side of the rectangle yields two rectangles, one with width 1280 x 5999/6000 and one with width 1280/6000. The latter rectangle is a sliver roughly 1/5 pixel wide and 1024 pixels tall. Of course it is possible to use approximate splits based on integer pixels or even to use three pixels so the resulting point will be inside a rectangle rather than on top of a one pixel width rectangle. While not all clustered datasets will produce slivers, slivers happen often enough to be troublesome. Why not partition so a corner of the rectangle contains the outlier? This question leads to the idea of partitioning lines at more angles than just multiples of 90 degrees and not limiting the partitioned area to rectangles, Sun and Carr (2005).

2.4 Spring Models

A graph *G* is defined as a set of vertices (or nodes) *V* connected by a set of edges (or links) *E*. Various algorithms have been developed to draw graphs (Di Battista 1998, Tamassia et al. 1988). The most widely known graph drawing techniques include spring-embedder algorithms and force-directed graph drawing algorithms (Eades 1984). The goal is to optimize the nodes arrangement in space such that strongly connected nodes appear close to each other, and weakly connected nodes appear far apart. This layout

process is a key topic in the graph drawing community. The strength of connections between nodes can be based on various similarity measures.

Network visualization is a major line of research in information visualization and graph drawing is an established field that investigates how to draw a network using the node-and-link representation in compliance with aesthetic criteria. Often large network needs to be pruned or divided into smaller parts to reduce its complexity before visualization. The pruning can be accomplished by reducing the number of links using minimum spanning trees or other methods.

The spring-embedder model was originally developed by Eades (1984) and now is one of the most widely used algorithms for drawing graphs. The idea is to connect each pair of nodes with a spring. The spring is associated with attractive forces (calculated only for neighboring nodes) and repulsive forces (calculated for all pairs of nodes), according to the distance or strength between the nodes' connections. The approach involves simulating pair-wise repulsive forces and attractive forces, pulling them closer together or pushing them further apart. This is repeated iteratively until the system reaches equilibrium.

Force-directed layout algorithm developed by Fruchterman and Reingold (1991) is a significant enhancement of the spring-embedder model using a different method to calculate the forces. Its unique feature is that all the nodes are moved together, making it possible to reach configurations that are not necessarily the current local minimum. A homogeneous model is commonly assumed. In such a model, the strengths of attractive and repulsive forces depend on the distances between nodes. They do not depend on

properties of individual nodes or weights of individual edges other than through global constants.

These algorithms have many advantages; one of them is that it requires no specific knowledge of graph's properties. They produce good quality results while being flexible, simple, intuitive and interactive. These algorithms are reliable general-purpose tools for graph layout applications and are widely used to obtain 2-D and 3-D layouts for undirected graphs.

Davidson and Harel (1996) describe how simulated annealing is applied to graph drawing. The simulated annealing model is oriented to the physical process of annealing, which often leads to very regular structures (e.g. like crystals). A global energy level is computed for a graph which is the sum of all energy levels of the nodes. The energy level at a node is determined from the forces acting on it, much like the elongation of the springs. The spring-embedder aims to minimize the global energy level by moving the nodes in the direction of the forces.

Simulated annealing is a flexible optimization method. It typically starts with a randomly chosen initial configuration and repeatedly reduces the value of a cost function. The algorithm differs from the standard greedy optimization methods by allowing 'uphill' moves. It is a necessary process to avoid being trapped at a local energy minimum in order to reach a global minimum. A key element is the choice of the cost function.

Sun, Smith and Caudell (2003) give a low complexity force-directed tree layout algorithm based on Lennard-Jones (LJ) potential that is widely used in computational

chemistry to simulate the interaction between two atoms or molecules. The recursive method follows the containment convention of drawing trees. It lays out each node of the tree as a disk. Each child node is contained inside its parent disk. Each node recursively contains all of its descendants. The children inside each node were positioned based on the total forces exerted on them. The layout looks like this:

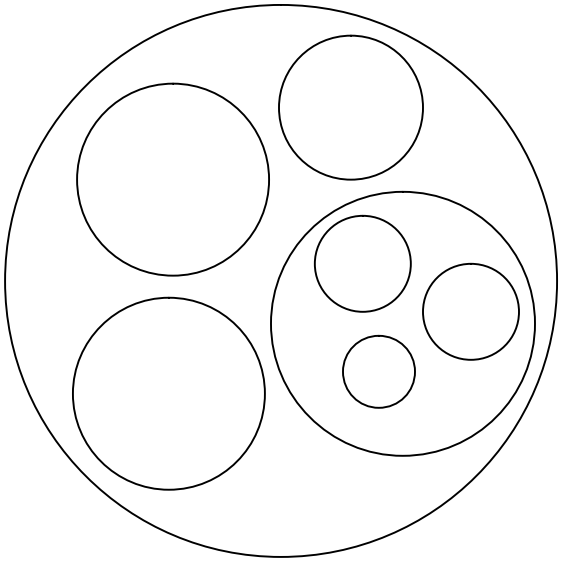

Figure 3. A Force-Directed Tree Example

This layout reflects both global structure and local details and supports run-time insertion and deletion. It is also visually appealing. The LJ potential plays the key role in avoiding overlaps. However, as circles do not form a tessellation of 2-D space, much space among the circles was not used. Also, it is not straightforward to define neighbors for circles.

Eick and Wills (1993) described an approach of placing nodes on the plane in the field of exploratory visualization of networks where there is a strength or weight associated with each link. Instead of using a pre-defined location, or using a layout purely to reduce clutter, they presented a method which positions nodes such that the distance between nodes is related to the strength of the link between them. Let d_{ij} be the displayed link length, and w_{ij} be the weight of the link; they elected to minimize the function

$$\sum (w_{ij} - \frac{1}{d_{ij}})^2 = \sum \frac{(d_{ij} w_{ij} - 1)^2}{d_{ij}^2}$$

This idea can be used in a layout problem. Once distances between clusters are defined, they can be treated as nodes and positioned on the plane, such that close clusters stay close. The same approach can be applied to sub-clusters as well.

2.5 Equally Spaced Rank Ordering

Seriation is a data analytic tool for finding a permutation or ordering of a set of objects using a data matrix (symmetric or asymmetric). The effect is to produce an equally spaced rank ordering. In the 19th century, archaeologists and prehistory researchers used seriation as a dating technique for ancient objects that are described using certain attributes. More recently, Marcotorchino (1991) discussed several aspects of the seriation problem including the problem setting, methodology and algorithms. There are two algorithms: constrained and unconstrained optimization. In constrained optimization, only rows are allowed to move. In unconstrained optimization, both rows and columns are free to move. Brower and Kyle (1988) describe a seriation algorithm.

Let the initial matrix be *T*, the set of objects be *I* and the set of variables be *J*. The basic principle of uni-dimensional seriation is to reshape *T*, with a permutation of *I* together with a permutation of *J*, such that there is a maximum density of high cell values along the diagonal of the resulting *T'*, in addition to low value areas in the upper and lower parts, or only the lower part of *T'*. This seriation process helps reveal the embedded latent structure. Bertin (1983) presents an example diagram. This method works well for small datasets. It could run into serious problems when applied to large datasets. Streng (1991) discusses classification and seriation by iterative reordering of a data matrix. Marcotorchino (1991) discusses block-seriation methods, the unified approach of block-seriation and its generalization.

2.6 Other Graphical Layouts

2.6.1 Hexagon Binning

Hexagon binning promoted by Carr et al. (1987) serves as a form of bivariate histogram to visualize the bivariate density structure in datasets with large *n*. In the data density context it serves to address point-overplotting problems and the use of bin summaries can speed up computations. Binning can also provide symbol congestion control in the mapping context and provide space for plotting glyphs. Carr et al. (1992). The underlying concept of hexagon binning is simple: 1) the plane is tessellated by a regular grid of hexagons; 2) the number of points falling in each hexagon is counted; 3) the hexagons are plotted using color classes or varying sizes in proportion to the counts.

The hexagonal bins are preferable over rectangular or square bins for their visual appeal and representational accuracy.

2.6.2 Tessellations

A tessellation or tiling of the plane is a collection of plane figures that fills the plane with no overlaps or gaps. One may also speak of tessellations of the parts of the plane or of other surfaces. Tessellation generalizes to higher dimensions. Tessellation is a basic algorithm in computational geometry with many applications. The most commonly used tessellations are Voronoi tessellation and Delaunay triangulation.

The Voronoi (or Dirichlet) tessellation is one of the most popular tilings in scientific graphics. Each pair of points is separated by a boundary based on the perpendicular bisector of the line segment joining the points. Voronoi tessellations enable the partitioning of an *m*-dimensional space without producing holes or overlaps.

The Delaunay triangulation is a dual graph of the Voronoi tessellation for the same set of points. Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid "sliver" triangles.

Balzer and Deussen (2005) developed Voronoi Treemaps to address the ‘sliver’ problem in treemaps. The algorithm starts with Voronoi tessellation of points within a fixed polygon. The points are iteratively moved and tessellated to make the areas of their containing Voronoi regions closer to the targeted relative areas until a convergence criterion is met. The algorithm can be re-applied to the individual fixed polygons produced by a Voronoi treemap to produce nested Voronoi treemaps. Voronoi Treemaps

are visually appealing. They can preserve the nested cluster property of Wills' algorithm. They are not conveniently structured to support reallocation based on the inclusion of white space to separate clusters.

2.6.3 Cone Trees

Cone tree was introduced by Robertson et al. (1991). It is a three-dimensional representation of hierarchical information. The hierarchy is presented in 3-D to maximize the effective use of available screen space and enable visualization of the whole structure. The node of the tree is located at the apex of the cone and its children are arranged around the circular base of the cone in 3-D. Each cone is shaded transparently so that it can be perceived yet not block the view of cones behind it. Moreover, cone trees provide dynamic views - any node can be brought to the front by clicking on it and rotating the tree.

2.6.4 Botanical Visualization

Kleiberg et al. (2001) propose a botanical visualization of huge hierarchical data structures. The method is based on a simple observation that people can easily see the branches, leaves and arrangement of a botanical tree in spite of the large number of these items. The algorithm is based on botanical modeling to convert an abstract tree into a geometric model. Non-leaf nodes are mapped to branches and child nodes to sub-branches. The Kleiberg et al. (2001) paper gives an example of laying out a directory

structure of 47,551 files and 2,265 folders. The botanical trees are innovative in design and visually intriguing, but their practical value remains to be seen.

2.6.5 Self Similar Layouts

Koike and Yoshihara (1993) describe fractal tree layout algorithm that addresses the problems associated with visualizing huge hierarchies. The geometrical characteristic of a fractal, self-similarity, allows users to visually interact with a huge tree in the same manner at every level of the tree. The fractal dimension is a measure of complexity that makes it possible to control the total amount of displayed nodes. Fractals are very useful in network evolution visualization. One of the most intriguing studies of the relationship between fractals and networks is due to Pickover (1988).

2.6.6 Space-Filling Curves

The subject of space-filling curves has generated a great deal of interest since the first such curve was discovered by Peano over a century ago. An *N*-dimensional space-filling curve is a continuous function from the unit interval [0,1] to the *N*-dimensional unit hypercube $[0,1]^N$. In particular, a 2-D space-filling curve is a continuous curve that passes through every point of the unit square. A space-filling curve is typically defined as the limit of a sequence of curves.

In a space-filling curve, a line passes through every point in a space in a particular order, according to an algorithm. The curve passes through points only once (i.e. the curve doesn't self-intersect), so each point lies a unique distance away from its beginning

along the curve. Thus a space-filling curve imposes a linear order of the cells in the *N*-dimensional space. This is useful in sorting of objects that lie in the multidimensional space.

There are many types of space-filling curves, e.g., Peano's, Sierpinski's, Hilbert's, Lebesgue's, Schoenberg's, etc. They differ from each other in the way they visit and cover the points in space. See Sagan (1994).

Wattenberg (2005) constructed a layout called jigsaw map combining the notion of space filling curves and treemaps. It satisfies four desirable properties of a layout function, namely nicely shaped regions, stability with regard to changing leaf values, stability with regard to changing tree structure, and preservation of ordering information. It maintains decent aspect ratios. However, the decoding of a jigsaw map is not straightforward and the appearance is not very visually appealing.

2.7 Discussion of Graphical Issues

2.7.1 2-D vs. 3-D

3-D display usually refers to 2-D perspective projections of 3-D environments. These 2-D projections may also include other pictorial depth cues, such as lighting model based shading of 3-D surfaces. The best-known example for 3-D visualization of trees is the cone tree of Robertson et al. (1991), refined and extended by others to display larger structures. The hope is that the extra dimension would give more space and a richer environment for cognition.

The first quantitative estimate of the benefits of 3-D stereo viewing for perceiving graphs was made by Ware and Franck (1996). They found that people performed better in 3-D than in 2-D. Ware and Mitchell (2008) conduct further experiments to study graph comprehension using a very high resolution stereoscopic display. The results show a much greater benefit for 3-D viewing than previous studies.

In Tavanti and Lind's paper (2001), they summarized the theoretical conclusions of some previous studies of 3-D displays, including Cone Trees and Data Mountain, as follows:

- 3-D displays can be exploited to visualize large sets of hierarchical data;
- The perspective nature of 3-D representations makes it possible to show more objects in a single screen, with objects shrinking along the dimension of depth;
- If more information is visible at the same time, users gain a global view of the data structure;
- There is experimental evidence that 3-D 'ecological' (more realistic and natural) layouts enhance subjects' spatial performances.

Tavanti and Lind (2001) performed two tests to compare the performances of 2-D display vs. 3-D display. The displays used in the experimental situation represented hierarchical information structures. The 2-D tree consisted of 27 rectangles, divided into four nodes and articulated into four levels of depth with dashed lines connecting the rectangles to signify the nested structure of the tree. The 3-D tree was composed of a window whose size corresponded to the visible part of the 2-D window. The tree also

consisted of 27 rectangles, divided into four nodes and articulated into four levels of depth. The difference is that the nested structure of the tree was expressed in terms of depth, so that higher levels were represented by larger rectangles while deeper levels were represented by smaller rectangles. The task is for subjects to click on the rectangles in any order to uncover all the characters associated with each rectangle and memorize as many characters' positions as possible. Two measures were used to evaluate the performance: The number of correct responses and the association of a character to the correct depth level in the tree. The test results showed that the 3-D display did improve performances in this designed spatial memory task.

2.7.2 Glyph

A glyph is a graphical object designed to represent multiple data values at once. To create a glyph, multiple data attributes are mapped in a systematic way to show the different aspects of the appearance of the graphical object, such as position, orientation, color, size, shape, texture or other features of a glyph (Ware, 2004).

Commonly used glyphs are Chernoff faces (Chernoff 1973), star plots, Exvis sticks, whisker plots, time series, histograms, profile plots, line-height plots, etc. For glyphs to be seen easily, they must stand out clearly from all other objects around them on at least one coding dimension. For example, a large symbol will stand out among small symbols, or a red symbol will be easily seen among yellow ones. Forsell (2005) explored and discussed the construction of 3-D glyphs for the visualization of spatial data.

The appropriate design and application of glyphs enables researchers to view the layout of the whole case set and zoom in to check the details of certain cases of interest. When possible, it is very important to have an overview of the cases. This allows the detection of overall patterns and aids in deciding the next move. A general heuristic of visualization design is to start with an overview and allow the users to access details as they choose. Shneiderman (1996) indicated, “overview first, zoom and filter, details on demand”. In the information visualization community, dynamic interactive linking and progressive disclosure are considered very important in exploratory data analysis and knowledge discovery. In the statistics community, overviews are often based on statistical summaries. My focus is on overview layouts that facilitate the inclusion of glyphs and links that lead to revealing object details. The glyphs or cells containing links may represent statistical summaries as opposed to individual cases.

2.7.3 Other Graphic Issues

Color is one of the effective attributes for associating categorical subdivisions with quantitative values. Color works equally well with points, lines and bars, as long as the object is not so small that users must strain to distinguish the colors. Appropriate use of color for data display allows relationships and patterns within the data to be easily observed. The careless use of color will obscure these patterns. Brewer (1994) described guidelines of color usage for mapping and visualization. There is also an online tool at www.colorbrewer.org for selecting specific color schemes.

Tufte (1983, 1990, 1997), Keller (1993), Grinstein and Levkowitz (1995) and Ware (2004) contain useful and interesting discussions on color usage, as well as other important graphical design issues.

Many other considerations are worthy being mentioned briefly here, such as the advantage of perceptual grouping, the ways to label the plots, the usage of gridlines, etc. Much literature can be found on these topics (Cleveland 1993, Cleveland and McGill 1984, Wilkinson 1999, Carr 1994a, 1994b, Kossyln 2006, etc.)

Chapter 3. Recursive Partitioning Layout in 2-D to Produce Round Regions

In this chapter I first introduce Wills' (1998) rectangular recursive partitioning algorithm for clusters. Then I extend this by adding gaps to visually separate clusters at each level of partitioning. The occurrence of very thin rectangles that I call 'slivers' motivates the Compact Area Partitioning Algorithm (CAP) that I developed to recursively partition a convex polygon into round polygons for containing the cases. The more compact, round polygons avoid slivers and can better accommodate glyphs or links to be used in tasks such as linked brushing. Additionally, I introduce the AIRS dataset and two special star glyphs to be plotted in the round regions. The methods are illustrated with examples.

3.1 Augmenting Treemaps Designed to Show Clusters

In section 2.3, I introduced Wills' rectangular cluster layout (Wills 1998) that motivated portions of this research. He recursively divides rectangular areas based on a cluster tree to provide 2-D plotting coordinates for points. Drawing rectangles and sets of points can show the clusters at any level of clustering without moving points. Wills shows an example with 40,000 points representing zip code regions. He indicates "the

linking both to and from the cluster view allows users to explore the clusters with respect to other variables and see if there are additional interesting dependencies." Figure 4 is an example of applying this algorithm to an ACE (Angeotensin–Converting Enzyme) Inhibitor dataset. The variables of the data are chemical properties of compounds and are coupled with measures of biological activity. This data was put together by Stan Young. The focus remains on the layout algorithm, so I omit further dataset details. After laying out the compounds, the placement of biologically active compounds in cluster nodes are highlighted in red. The cluster view shows that the active compounds concentrated in a few clusters. The encoding of biological activity turns the plotted points into the most elementary form of a glyph. The study of a linked scatterplot matrices that highlights the properties of active compounds has the potential to shed insight into the property boundaries between active and inactive enzymes.

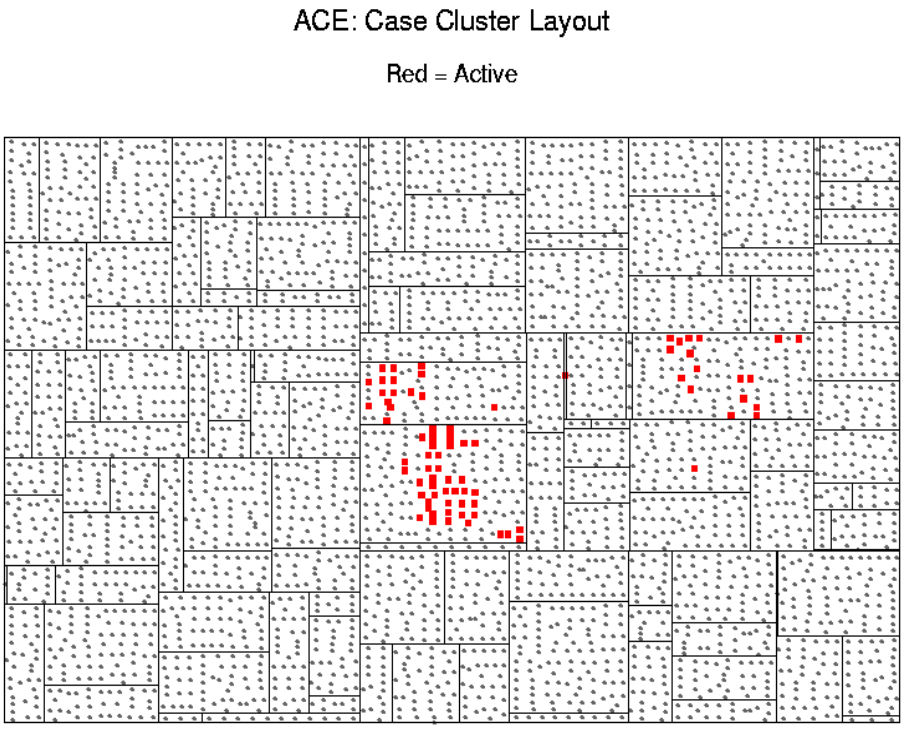

Figure 4. Rectangular Recursive Partitioning of ACE Data

When making the splits, white space (gaps) can be added between sub-areas to show the separations. The width of the gap can be a monotonic function of the dissimilarity between the two sub-clusters. This helps researchers see how far sub-clusters are apart. The separation "distances" is typically secondary to showing hierarchical cluster membership. I limit the gaps to two or three distinguishable sizes. Factor of 2 size differences are easily distinguished (Kosslyn 1994), thus there are distinctive separations rather than continuous separations. When the distance falls below a threshold I stop showing separations. However, the partitioning continues until there is only one case in each rectangle. The centers of the rectangles provide location to plot

points or glyphs. This partitioning procedure allows lines to be drawn at any level of clustering without moving the points.

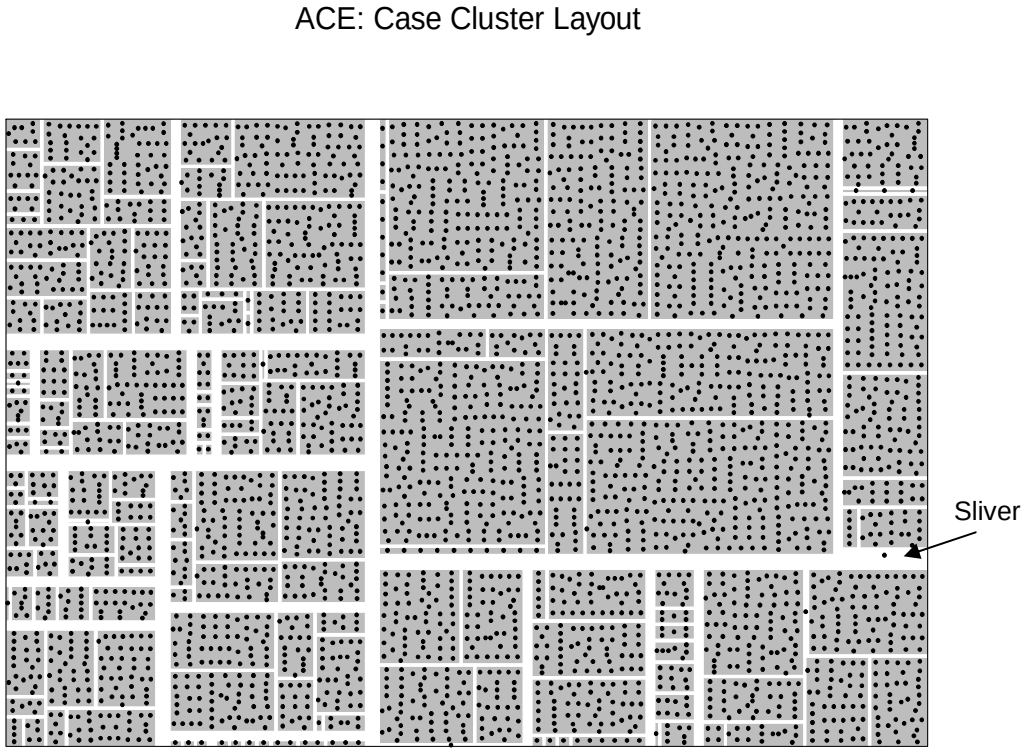

Figure 5. The Rectangular Layout with Gaps

Figure 5 shows the rectangular layout with gaps added. The first cut is the thick vertical white line. The second cut is a thick white horizontal line on the right. There is a 'sliver' at the bottom right of the top right rectangle created by these two white lines. The rectangular region is too thin to show in gray, so a single dot is plotted as labeled in Figure 5. There are other cases of 'long-thin' areas, for example, the vertical area to the

right of the first cut at the top of the plot, and the horizontal area at the bottom left. The example at the end of section 2.3 discusses an extreme outlier situation.

The subtraction of area for white space can alter the area available to plot points and make the point density less uniform. A more sophisticated algorithm could cut rectangles into three pieces, allocating white space rectangle in the middle of the two rectangles for sub-clusters. Iteration could refine this by adjusting the relative sizes of the rectangles for sub-clusters based on the amount of white space they contained in the previous iteration.

A 3-D lighting model can be applied to create 3-D effects to further visually convey the idea of separation. Carr and Sun (1999) discuss the usage of a lighting model in plots to add value to the appearance. The lines can be changed into grooves with different depth and a lighting model applied to make the gray rectangles appear on plateaus of different heights. See Figure 6 below. Getting the groove intersections to look exactly right requires some experimentation and better views are possible.

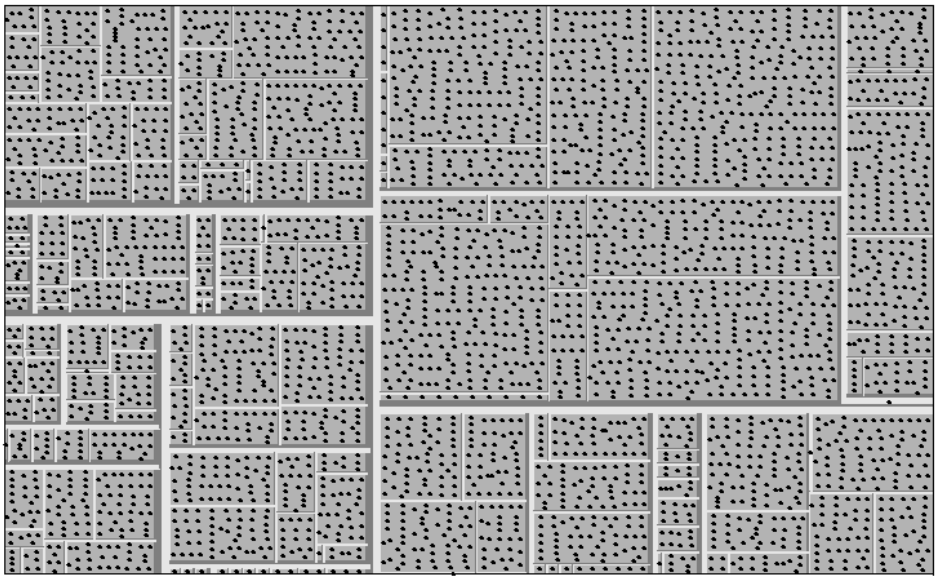

Figure 6. The Rectangular Layout with 3-D Gaps

3.2 Compact Area Partitioning (CAP) Algorithm

To produce more compact polygons to surround the cases and rounder regions for glyphs and links, I introduce three extensions to Wills' algorithm.

1. I provide the option to start with a plotting region other than a rectangle. Starting with a rectangle seems generally preferable for optimal use of a rectangular display space but flexibility suggests being able to start with a more compact polygon such as a hexagon.

2. I use partitioning lines at multiples of 60 degrees as well as multiples of 90 degrees. The 60 degree matches the angles used in hexagons. Since the region being partitioned is a convex polygon, more bookkeeping is required than with rectangular regions. I use iterative bisection method to locate the position of a cutting line that partitions the polygon into two sub-polygons with the areas proportional to the number of cases in the sub-clusters. The areas are calculated using determinants. Note that after the first cut, the convex polygon to be partitioned in each step is rarely symmetric. Consequently, for each cutting direction, there are two cutting positions that yield areas with suitable relative sizes. I calculate both cutting positions for each direction. The number of options to choose from is twice the number of unique cutting directions.

3. To determine which pair of sub-polygons to select in terms of roundness, I calculate the dimensionless second moment about the polygon centroids (Conway and Sloane 1999) and pick the pair with the smallest sum of dimensionless second moments. The criterion borrowed from coding theory provides a measure of "roundness" with circles and spheres having the smallest values in 2-D and 3-D respectively.

To avoid getting infinity from tangent of right angles and for calculation simplicity, I rotate the hexagon by 15 degrees. I consider cutting lines from six directions. Three directions are the same as the sides of the hexagon, and the other three directions are perpendicular to the lines connecting the center and hexagon vertices. This can be

generalized to any convex polygon and provides the option of cutting a corner of the polygon to form a ‘rounder’ area than a long, thin rectangle.

Figure 7. The Recursive Partitioning of a Hexagon

Figure 7 shows the results of applying the CAP algorithm to the same ACE dataset starting with a hexagon. The red lines indicate the levels of clustering. The sliver that appeared in the previous rectangular layout is assigned a triangle corner in this new layout, which is a ‘rounder’ area to accommodate a glyph or a link. The result somewhat looks like a city map and I find this a more aesthetically pleasing layout than rectangles.

I presented the algorithm and result at the Joint Statistical Meetings in 2005 (Sun and Carr 2005). We conjecture that the close points tend to be better and more uniformly separated than using a rectangular layout but do not have a proof. If our conjecture is true, the layout would be preferable for drawing round star glyphs at each point. To the extent that the CAP algorithm avoids slivers, it is clearly preferable.

There can be problems with the visualization of algorithmic results and algorithm speed. All algorithms will fail in terms of direct visualization when there are too many points for the available plotting space. S-PLUS and R scripts take a little time to run. The current implementation in C is very fast.

3.3 An Multivariate Atmospheric Data Example with Enhanced Star Glyphs

This layout-with-glyphs example uses cluster summaries produced from AIRS data. The AIRS data refers to the massive dataset that is collected by the Atmospheric Infrared Sounder (AIRS) which is an instrument onboard the polar-orbiting Earth Observing System Aqua satellite. The AIRS instrument is a high resolution spectrometer with 2,378 bands in the thermal infrared spectral region (3.74 - 15.4 micrometers) and 4 bands in the visible spectral region (0.4 - 1.0 micrometers). It measures Earth's atmospheric parameters around the globe. All data are released to the public. "Currently researchers are using AIRS data products to validate climate models and to test their representations of critical climate feedbacks." (http://daac.gsfc.nasa.gov/AIRS).

The Level 1 processing of this data handles geospatial and radiometric calibration. Level 2 processing converts this to geophysical parameters such as temperature, water vapor and cloud fraction. Level 3 processing produces summaries on equal angle grids of the earth. Many terabytes of data are involved.

This data was formerly studied by Amy Braverman and Eric Fetzer (Braverman and Fetzer 2006) from Jet Propulsion Laboratory (JPL), California Institute of Technology. They used the data for each 5^{o} x 5^{o} latitude and longitude cells of the earth and applied an Entropy-Constrained Vector Quantization (ECVQ) algorithm to produce multiple multivariate multi-altitude cluster summaries for each grid cell. ECVQ can be understood as a clustering algorithm similar to K-means algorithm that includes an entropy constraint allowing it to provide fewer clusters when a quality criterion is met.

Each cell contains 240 granules per day; each granule is 90 x 45 footprints; and each footprint is associated with one 35-dimensional data vector with the following components:

- Atmospheric temperature at 11 vertical levels (1-11),
- Atmospheric water vapor at 11 vertical levels (1-11),
- Cloud fraction at 10 vertical levels (1-10),
- Land fraction of footprint,
- Day/night flag, and
- Quality flag.

The input to ECVQ is all 35-dimensional data vectors with footprints inside each cell, and the output from ECVQ is a set of representative vectors of clusters and their

associated weights and distortions (average squared distance between original and representative vectors).

The earth equal angle grid consists of 5^o x 5^o cells, and there are 36 latitude rows and 72 longitude columns. The data used in the following example is a subset of the 2002 winter data, where latitude grid cell numbers are from 18 to 23, and longitude grid cell numbers are from 60 to 65, thus it covers a 30^o x 30^o area. The data includes 382 cases, where each case is an ECVQ output representative vector of one cluster within a grid cell. This particular summary for 36 grid cells has an average of a little over 10 clusters per cell. I selected the following variables for the representation using glyphs:

- Temperature at 11 vertical levels,
- Water vapor at 11 vertical levels,
- Cloud fraction at 9 vertical levels,
- Land fraction, and
- Day/night flag.

I exclude the cloud fraction at level 2, since almost all of the values for this variable are zero. The variable ‘quality flag’ is also excluded. Figure 8, 9 and 10 show boxplots of the distributions of the altitude specific variables.

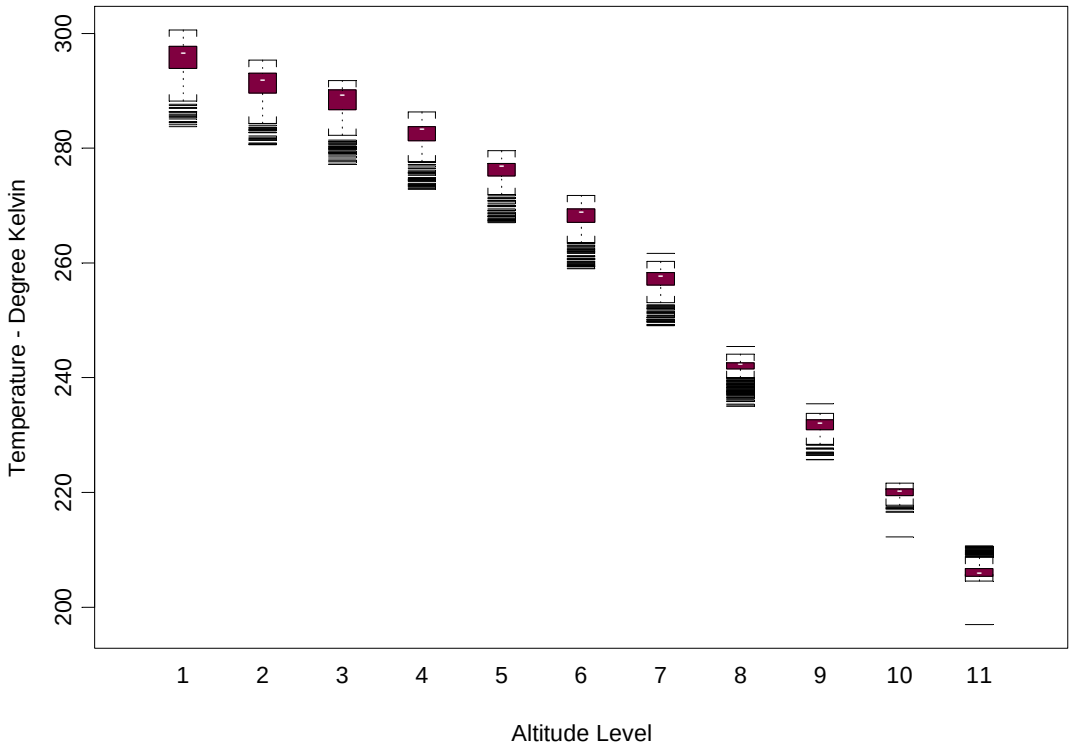

Figure 8. Boxplots for Temperature Variables

The median temperatures decrease as the altitude level increases. Except for the highest altitude, most outliers are associated with lower temperatures. The inter-quartile ranges (IQR) shown in color tend to be distinct for the different altitudes, although a little overlap between altitude levels 2 and 3.

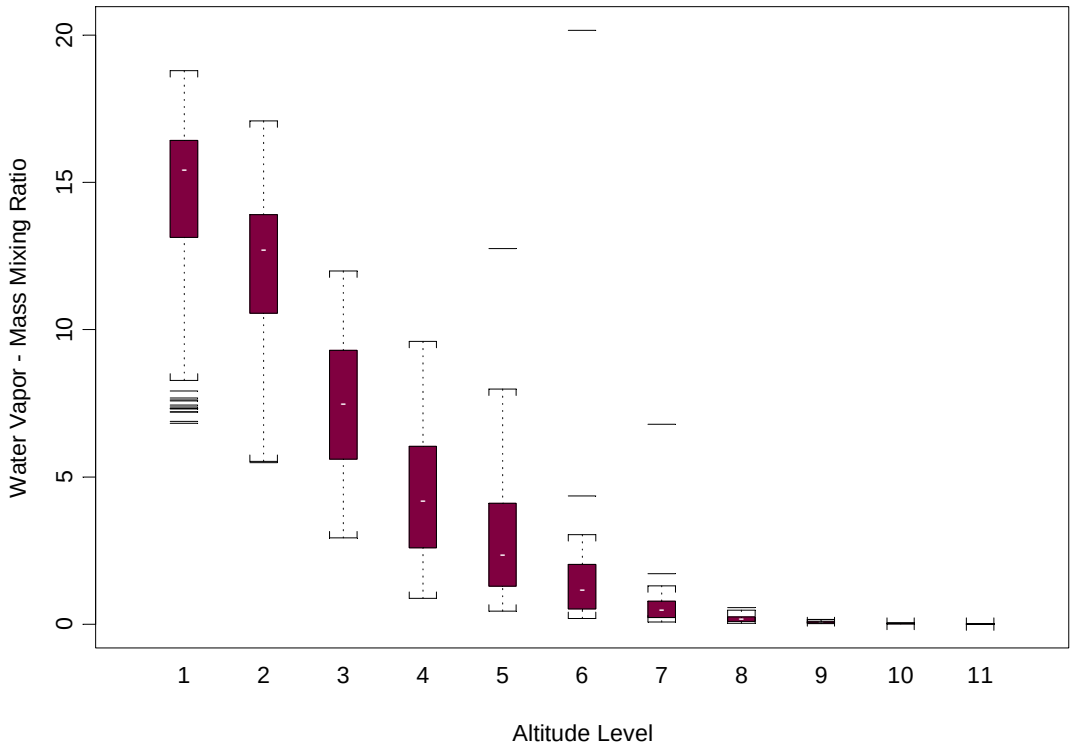

Figure 9. Boxplots for Water Vapor Variables

Similar to the temperature variables, the median water vapor values decrease as the altitude level increases. Water vapor values at altitude levels of 9, 10 and 11 are close to zero.

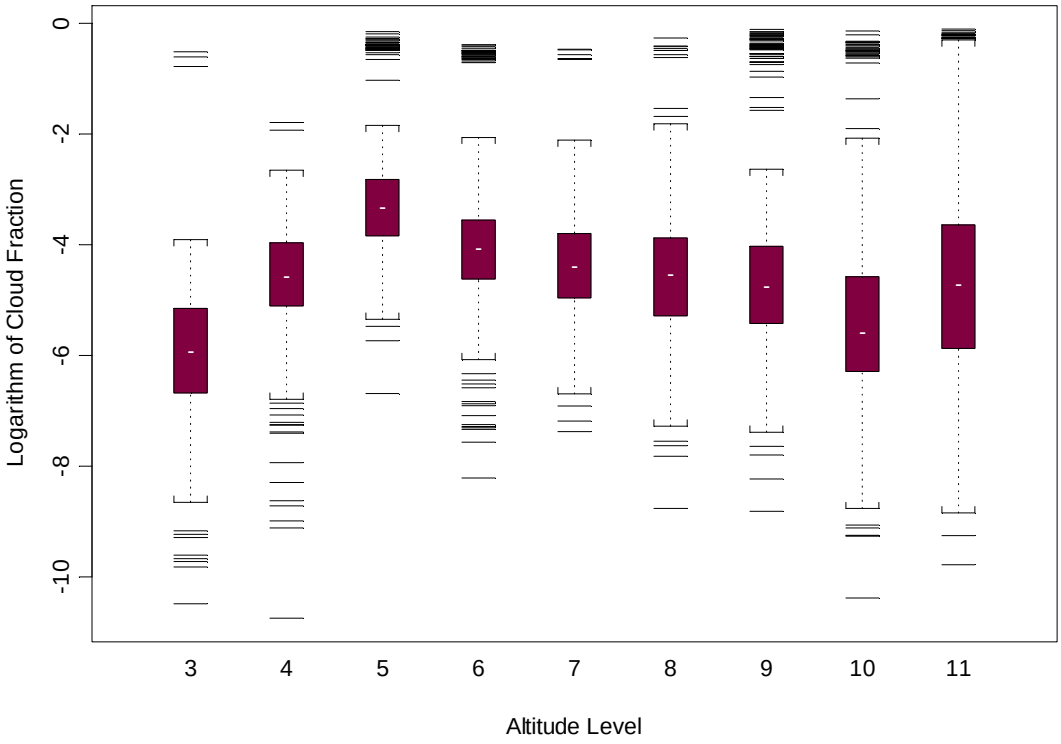

Figure 10. Boxplots for the Logarithm of Cloud Fraction Variables

Since there are many high values for the cloud fraction distributions, I use the logarithm of the cloud fraction in the boxplots to provide a more symmetric appearance.

I cluster the multiple vector summaries (cases) for the grid cells to provide a cluster hierarchy for the layout of these cases. The clustering involves scaling the 33 variables to have mean 0 and standard deviation 1. The agglomerative clustering algorithm uses the Euclidean distance among the scaled variables. Figure 11 shows the result of applying the CAP algorithm to this hierarchical structure. The layout differs from the layout of Figure 7 in that it shows all the final polygons containing a single case.

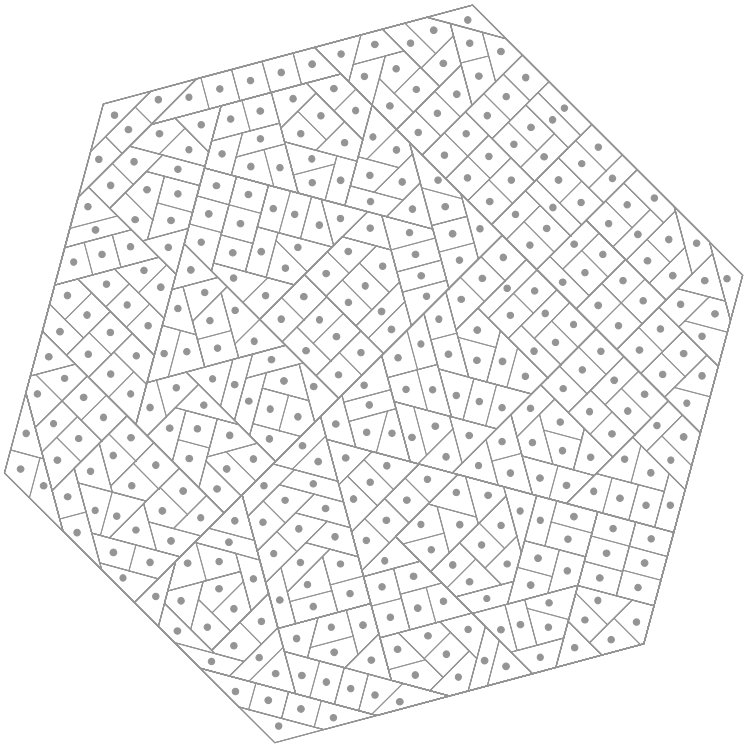

Figure 11. Round Polygon Layout for AIRS Data

The goal of showing many variables with a symmetric encoding using this layout motivates the development of new glyphs. I begin the discussion of new star glyphs with related glyphs for rectangular regions. The beginning glyph of interest has informally been called a comb glyph. It is basically a bar plot with lines as bars and a line that connects the base of the bars. Carr et al. (2000) discuss this as a line height plot and use it to show 159 variables. Importantly they both sort the variables to simplify appearance and add grid lines to improve the accuracy of assessing line length based on Weber's Law (Cleveland 1994). (For AIRS data the variables are ordered by altitude level, so

reordering the variables has less merit in this setting.) When there are fewer and more widely spaced lines, the line height glyph can be converted into a profile glyph. Figure 12 illustrates the general idea. Profile glyphs for a small number of variables can fit within the rectangular portions of a hexagon layout. In general, it seems more natural to put round glyphs (e.g., star glyphs) in round polygons, such as hexagons or the polygons of Figure 11.

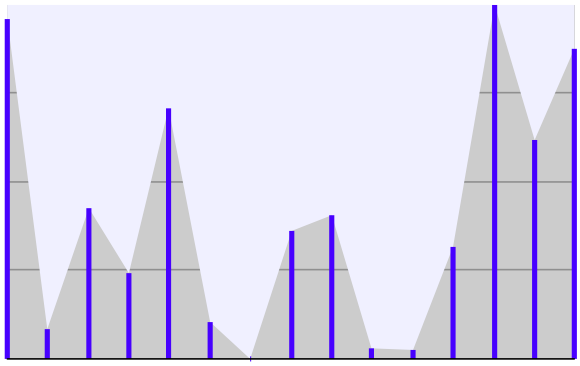

Figure 12. A Profile Glyph Example for 15 Variables

Many variants of the star glyph also use line lengths to encode values. The lines radiate out of a center point and typically appear at equally spaced angles. Figure 13 shows an example.

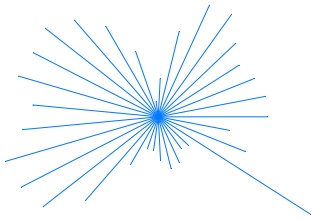

Figure 13. A Star Plot Glyph Example

The common scaling for comb and profile glyphs linearly scales the variable range into the interval 0 to 1. The lines with longer length tend to draw more visual attention than very short lines, including those of length 0 that encodes the minimum value of a variable. Readers may even wonder if there is missing value. The star glyph with many lines has an additional overplotting problem that can hide short lines near the origin.

I create two variations of star glyphs that provide a more symmetric treatment of small and large values with both being encoded using long lines. The designs use short lines to show values in the middle of the distribution, so the middle of the distribution gets less visual attention and some patterns can be hidden by overplotting. Thus these two glyph variations are primarily of benefit for observing both small and large values and for the appeal of symmetry. Figure 14 and 15 show the two new glyph variations called ‘signed deviation’ and ‘folded deviation’ star glyphs respectively.

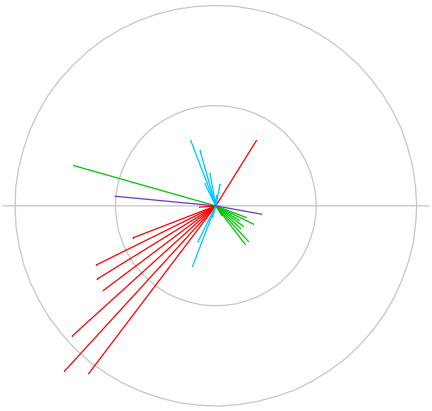

Figure 14. Signed Deviation Star Glyph

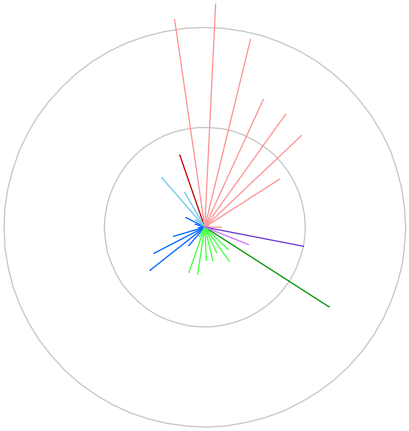

Figure 15. Folded Deviation Star Glyph

Both designs use reference circles in the background as a carry over from the use of grid lines in line height plots as mentioned above. For both glyphs, the common encoding first transforms the variables to have mean 0 and variance 1, and the reference

circles are drawn for 1 and 2 standard deviations. The glyph size is a scale factor chosen so that the line lengths are in a reasonable range to stay mostly within the polygons and not overplot on the other glyphs too much. Plotting dots on the 1 standard deviation reference circle can help with the alignment to identify variables that are displayed as very short or missing rays.

Other transformations can be chosen to address specific tasks. For example, a power transformation preceding the glyph scaling can provide a more symmetric distribution. Robust transformation based on the median and inter-quartile range (IQR) may be used to reduce the impact of outliers on the transformation. The extreme values can be Winsorized to allow a larger glyph scale factor. The Winsorized values can be highlighted by plotting dots at the line ends to draw attention. Rather than transforming variables individually, the transformation may be applied to composite of all variables with the same unit of measure. Such star glyphs would primarily show altitude level effects on temperature rather that the deviations within a given altitude. The same applies to the variables of water vapor and cloud fraction.

The design for the signed deviation star glyph in Figure 14 generates $p+2$ angles from 0 to 180 degrees, where p is the number of variables. It uses the angle of 0 and 180 to form a horizontal reference line. Carr et al. (1992) find angular grid lines helpful in decoding bivariate ray glyphs more accurately where values (as opposed to variable identity) are encoded using angles. Here the horizontal lines help distinguish between variables with positive and negative values. Trosset (2005a) uses an angular encoding that goes in both directions.

The rays go counter clockwise starting from the 3 o'clock position. Red rays are the eleven temperature variables; blue rays are the eleven water vapor variables; green rays are the nine cloud fractions; the remaining two purple rays are land fraction and day/night flag. The values above means are shown above the horizontal line, and values below their mean are shown below the line following the corresponding direction. For example, the red lines on the left side above the reference line are the temperature variables from high altitudes, while for negative scaled values shown below the reference line, the red lines on the left actually correspond to the right side above the line had their values being positive, thus represent variables from low altitudes. This involves more mental effort to identify variables, and motivates the second design.

The design for the folded deviation star glyph in Figure 15 generates $p+1$ angles from 0 to 360 degrees, where p is the number of variables. Same as in the signed deviation star glyph, color encodes variable categories. Within each variable group, the glyph uses two levels of the group's color to encode the positive and the negative values - the dark color represents positive values and the light color shows negative values. The length of the ray for each variable is the same as in the signed deviation star glyph.

Figure 16 is the round polygon layout of AIRS data with the signed deviation star glyphs. The glyphs are allowed to overplot a little to improve the resolution. We can see the partitioning of the groups and that glyphs in the same group have similar appearances. This can serve as a visual validation of the clustering algorithm and we can identify outliers at the corners of the area, as indicated. The highlighted portion was enlarged and shown in Figure 18.

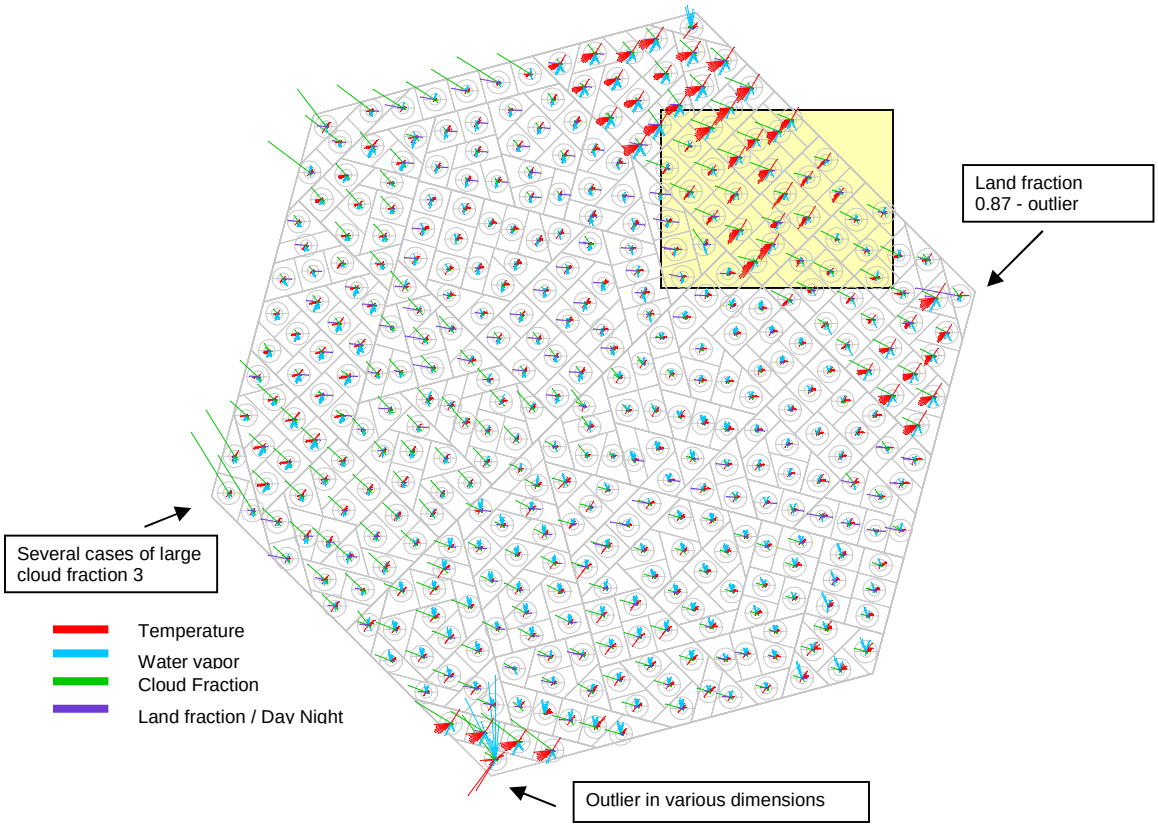

Figure 16. Round Polygon Layout with Signed Deviation Star Glyphs

The outliers at the high end of the water vapor variables shown in the boxplots in Figure 9 belong to the case located at the bottom of this layout and marked with ‘Outlier in various dimensions’. The three outliers in variable ‘cloud fraction 3’ shown in the boxplot in Figure 10 were put in a corner at the left side of Figure 16. Many long red rays are below the horizontal line, which is expected because the boxplots in Figure 9 show that most outliers are associated with low temperatures. By linking the overview layout in

Figure 16 with other statistical plots or summaries, users can have a better understanding of the data.

The longer cutting lines in Figure 16 help to reveal clusters. Figure 17 shows the same round polygon layout without the separating lines. Some clusters are evident from the star glyphs.

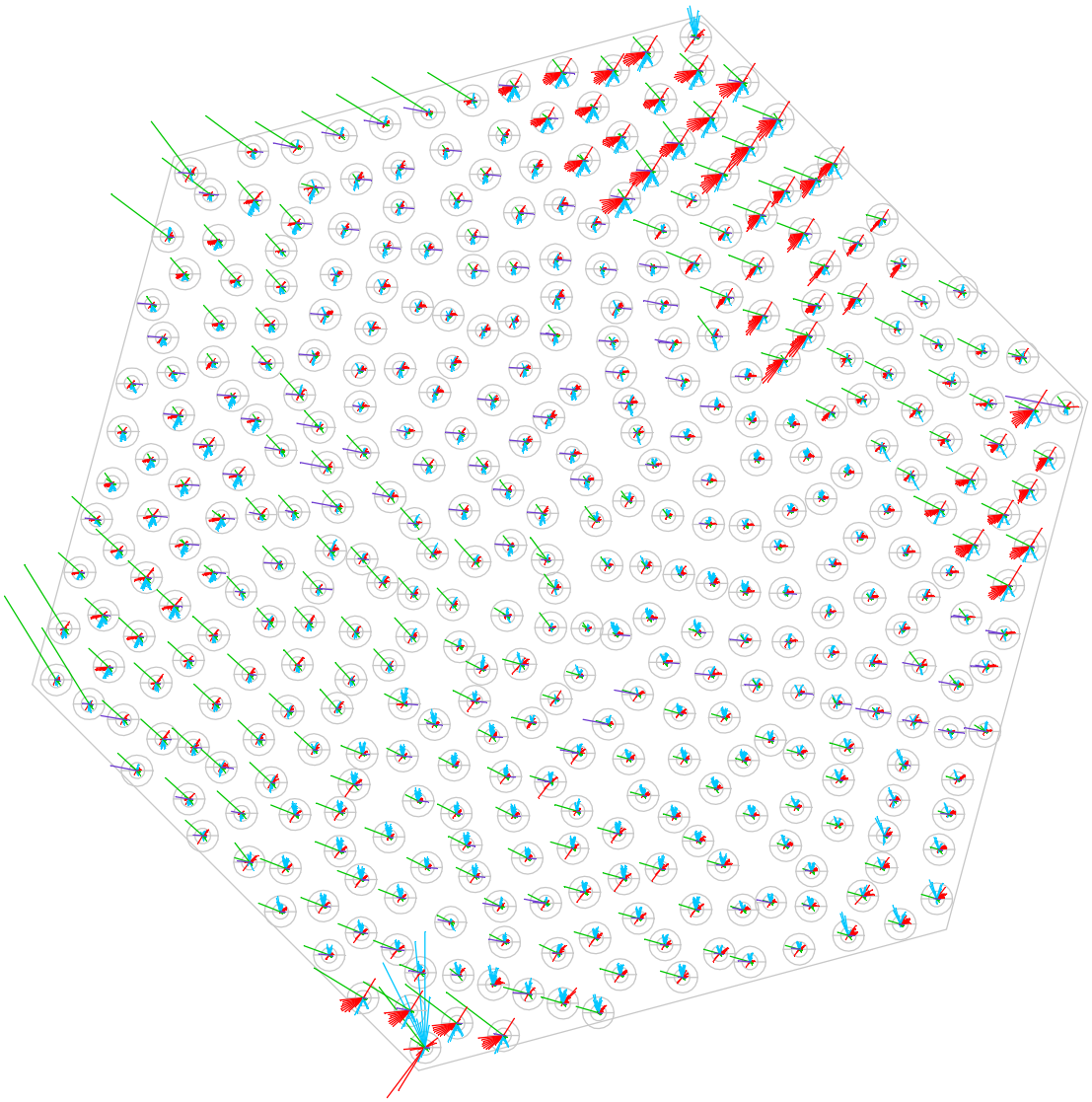

Figure 17. Round Polygon Layout with Signed Deviation Star Glyphs Omitting Separators

The plot barely shows green lines in the negative directions. This is because there are very many cloud fraction values near zero, whose deviations from the mean are negative and small compared to the variations of larger values. Taking the logarithm

transformation used in Figure 10 would reduce the variation of larger values and reveal more of the variation in the small values.

These overview layouts can be zoomed in to further reveal the details of the cases. Figure 18 is a zoomed-in view of the highlighted part of Figure 16. This supports better viewing of individual cases.

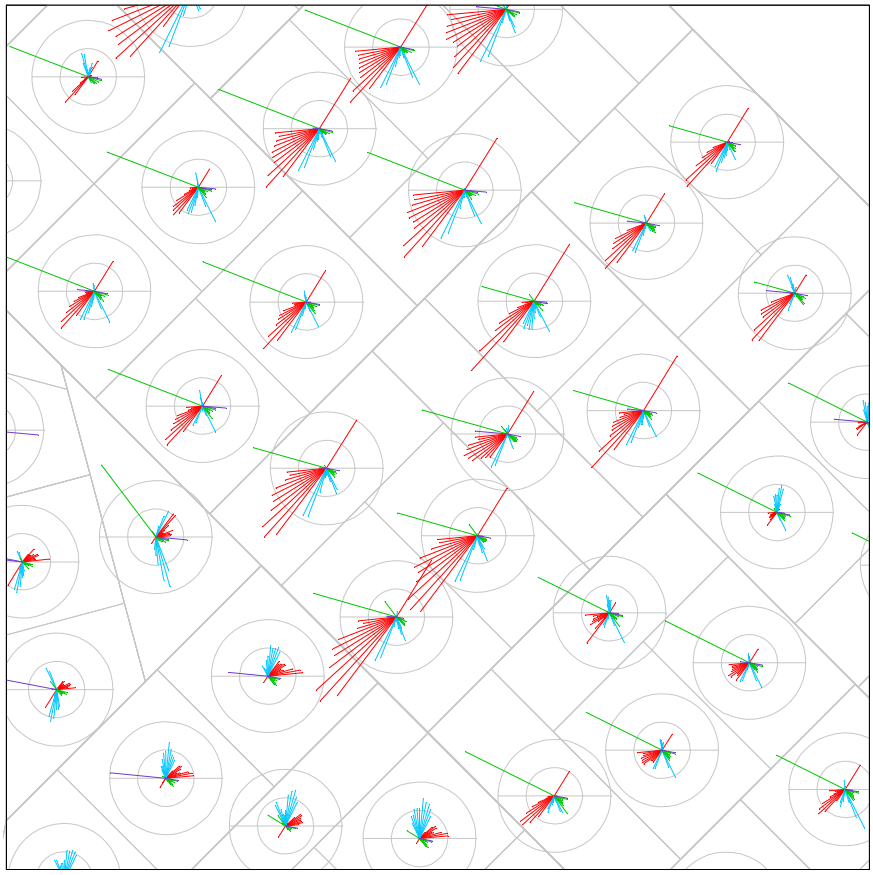

Figure 18. Enlarged View of Polygon Layout with Signed Deviation Star Glyphs

Figure 19 shows a round polygon layout with the folded deviation star glyphs. Note that the long red rays below the horizontal line in Figure 16 indicating small temperatures values are now displayed as light red rays.

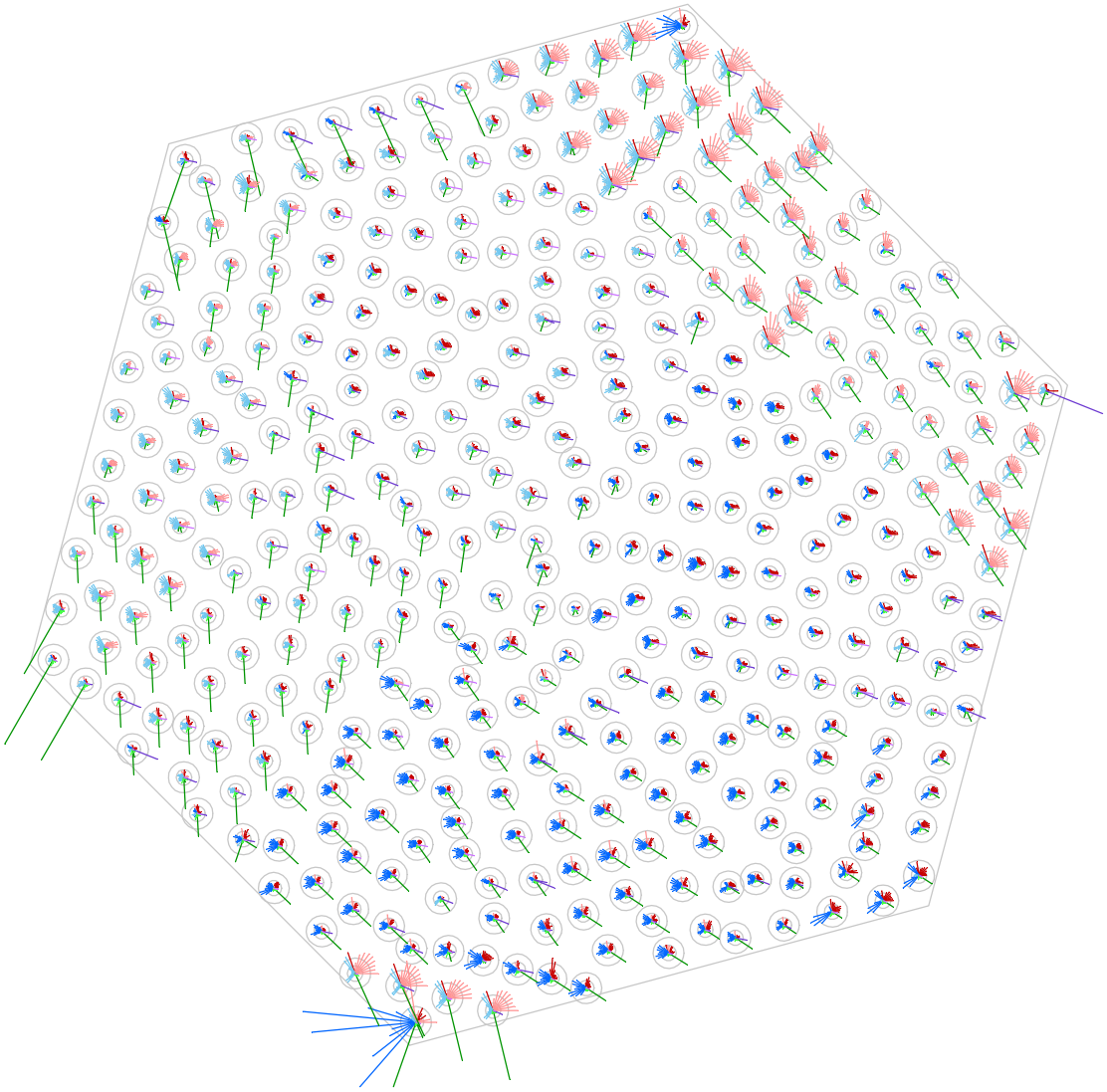

Figure 19. Round Polygon Layout with Folded Deviation Star Glyphs

Figure 20 shows the zoomed-in view of the same cases as in Figure 18.

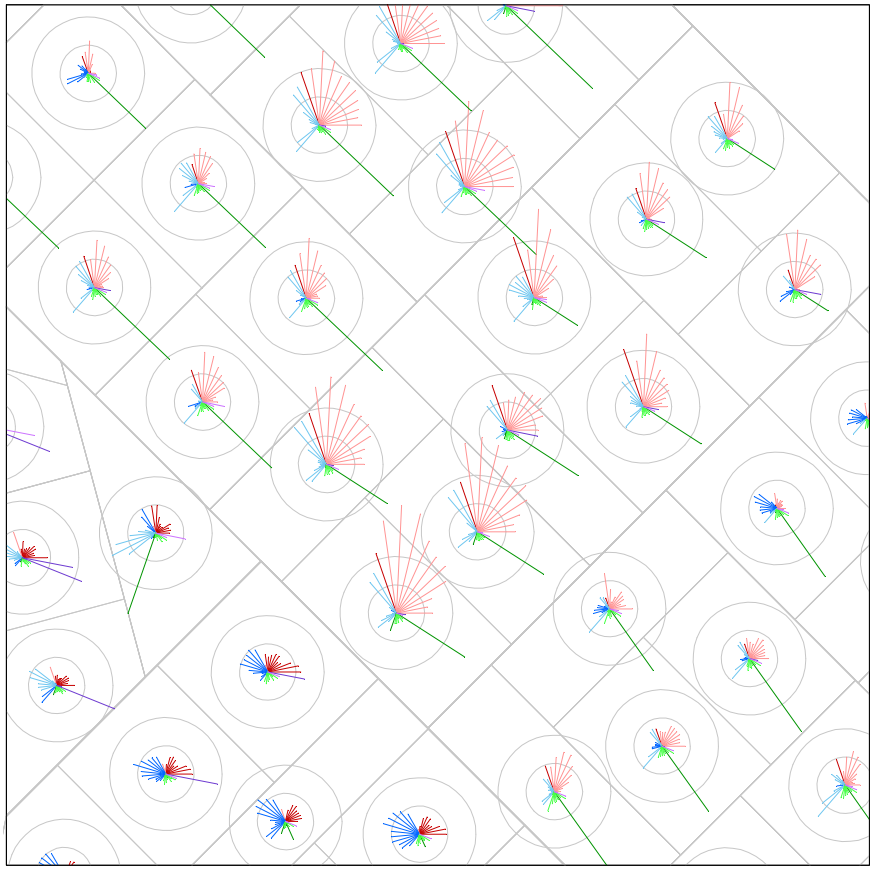

Figure 20. Enlarged View of Polygon Layout with Folded Deviation Star Glyphs

3.4 Summary and Assessment

This chapter addresses the layouts of hierarchical structures as nested polygons and the use of glyphs to show multivariate cases in the polygons at the bottom of the hierarchical structure. The enhanced rectangular recursive partitioning layout shows the separations between sub-clusters better by adding white space between the two sub-areas.

The round polygon layout produced by the CAP algorithm addresses the 'sliver' problem caused by outliers that often appears in the rectangular layout by allocating 'rounder' areas, so that they can better accommodate the glyphs or links associated with the cases. The layout is useful for an overview of all the cases. The signed deviation and folded deviation star glyphs provide a more symmetric encoding of large and small values than traditional star glyphs. The glyphs help to reveal the variation in case variables within and across clusters. The reference circles help to make more accurate comparisons. The layouts and glyphs were illustrated with examples.

The CAP algorithm was produced in the year 2000-2001 time frame. Since then a close competitor has emerged with merits of its own in terms of region and shape. This is the Voronoi Treemap layout (Balzer and Deussen 2005). It is possible to produce a nested cluster layout by repeated use of their algorithm. For a binary tree, the first layout produces two polygons with cluster areas of appropriate sizes for the two sub-trees. The algorithm is then applied iteratively to these polygons until polygons are created for the leaf nodes.

The appearances of the layouts can be compared, but preference may be more personal opinion than objective. I like fewer angles for the polygon edges.

Both methods address the area of polygons, both produced round regions. The CAP algorithm makes it better suited for introducing white space between clusters. Neither guarantees the regularity of hexagons that is advantageous for plotting. Hence I turned to the development of algorithms for hexagons that provide an option when the irregular regions are deemed less desirable.

Chapter 4. Point Layout for Hexagon Grids

This chapter addresses the layout of cases in a hexagon grid with white space. Carr (1991) discusses the merits of lattices where the cells are hexagons in 2-D, truncated octahedrons in 3-D and 24-cells in 4-D. Hexagons and truncated octahedrons are polygons and polytopes with three important properties:

- They can tessellate the respective space (2-D, 3-D),
- They are the closest to being circular and spherical respectively of all single polytopes in the tessellation space, and
- They have a well-defined number of neighbors (6 and 14 respectively).

In addition, although many algorithms use square (2-D) or cube bins (3-D) due to their simplicity in computing, our visual system often works better with hexagons and truncated octahedrons. In 2-D human beings have a strong reaction to horizontal and vertical lines. When such lines are construction artifacts, as in the binning context, they are distracting. Either reduced-size square glyphs that encode bin counts or round glyphs in square cells accentuate the two sets of orthogonal lattice lines, typically oriented to be horizontal and vertical. Hexagons with three non-orthogonal lines through opposite neighbors help the data reveal itself by reducing emphasis on orthogonal visual lines.

Hexagon binning has been a part of S-PLUS for years and is currently being integrated in R within the lattice package. New algorithms for other uses of hexagon grids can expand the varieties of applications.

The basic task of the hexagon grid algorithms here is to place individual cases in separate hexagon cells so that similar cases are close together. There are two helpful distinctions. The first distinguishes between hexagon grids with more internal cells than cases with the locations of the empty cells subject to change as well as the cases (cell occupants), and occupied subsets of hexagon cells where only the cell-occupied cases are exchanged. The second distinguishes between the layout of clusters and the layout of cases based on similarity. Both can work using a dissimilarity matrix. The cluster dissimilarity can ignore details other than cluster membership. Dissimilarity for a generic empty case (occupant of an empty cell) can be added to the dissimilarity matrix so the case exchange algorithms can address the empty cell case without modification.

The algorithms have a cost function based on the dissimilarity of each case to its immediate neighbors, thus local relationships drive the layout of cases. This contrasts to algorithms such as classical multidimensional scaling, in which cases very far away from the body of data can force cases of well separated in the body of the data to be close together in the layout.

I present the research sequence starting with a simple and perhaps naïve initial layout algorithm, and progress to algorithms that take control of the initial layout and then refine the placement of cases.

4.1 A Preliminary Hexagon Occupant Exchange Algorithm (HOE)

Hexagon case exchange algorithms are not new, Wills (1999). The variation investigated here addresses the exchange in the present of empty cells. Carr wrote the first Hexagon Occupant Exchange (HOE) Fortran program used in this research. The algorithm addresses the situation where roughly half of the cells are empty after the cases are randomly loaded into a hexagon grid. There are two types of cells: occupied cells that are attached to individual cases and empty cell that is not attached to cases. The idea is to make the empty cells ‘flow’ between clusters.

The basic algorithm uses a congruential random number generator to randomly pick two cells in the grid. The cases occupying these cells are swapped when the exchange reduces a cost function of the twelve distances to the cases in the neighboring cells. The iteration stops when the exchange no longer reduces the cost for several cycles. The cost function used has two variants. The fastest version sums the twelve distances of the two cases in the chosen cells with the cases in the immediate neighboring cells. The slow one sums the two distances to the fourth closest case among the immediate neighboring cells. Most of the study centers on the faster algorithm.

The distances between cases in two occupied cells are the pre-calculated dissimilarities between the cases. The distance between a generic empty case in an empty cell and a case in an occupied cell and the distance between two generic empty cases are two free parameters. My research concerns picking two percentiles of the distribution of dissimilarities to use for the two parameters. When distinct clusters were not always

separated, Carr added a simulated annealing option. I was able to find percentiles that worked for a few examples. Figure 21 shows one.

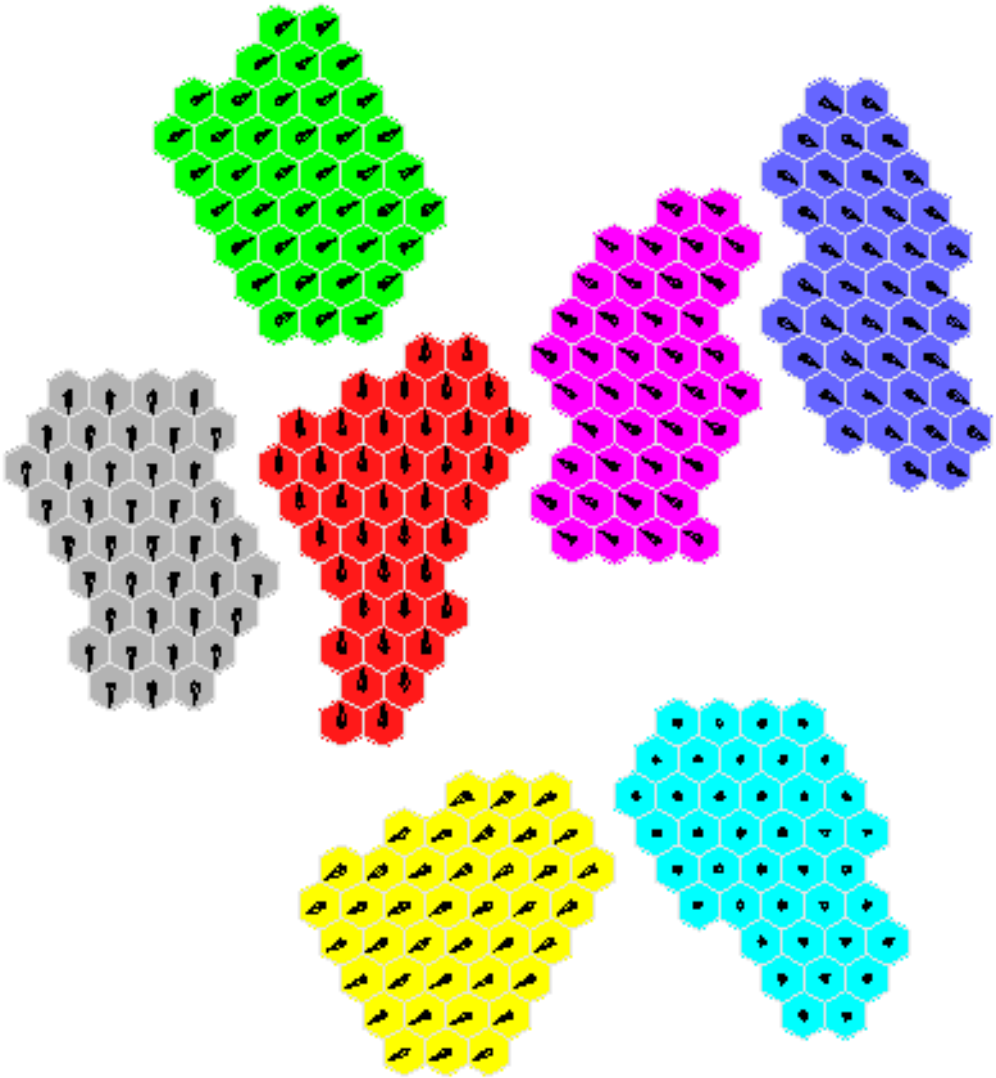

Figure 21. Cluster Layout on Multivariate Data in 6 Dimensions

The generated data in this example consists of clusters at the vertices of a simplex in six dimensions. The data generation idea is from Friedman (1977). One vertex

is at the origin and the other simplex vertices are six standard deviations away in the positive direction on each of the six axes. Random values from a standard normal distribution are added to vector coordinates of the vertices. There are seven distinct clusters in the data.

The backgrounds of the occupied cells in the layout are drawn in different colors to reflect cluster membership. With an unknown data structure, the color could encode group information determined by a cluster algorithm or natural/known categories. The clusters are separated in the layout. The empty cells visually emphasize local cluster edges. The cyan cells with dots show the points around the origin. The ray angles provide a clockwise encoding indicating the positive simplex coordinates. For example, the rays pointing straight up indicate that the first coordinate was positive. I gave a presentation on this algorithm at the Joint Statistical Meetings in 2000 and it was well received.

4.2 The Current Hexagon Occupant Exchange Algorithm

The current HOE algorithm is implemented in R. In the current version of the program, another type of cell is added besides the ‘Occupied cells’ and ‘Empty cells’, namely Border cells that locate at the border of the hexagon frame. As a result, two additional parameters need to be specified, the distance between a generic border case (occupant of a border cell) and a case in an occupied cell, and the distance between a generic border case and a generic empty case. The new algorithm drops the simulated annealing that was in the original Fortran program.

The cell indices of a hexagon frame are specified sequentially from the bottom left going left to right and then row by row until it reaches the upper right, as shown in Figure 22.

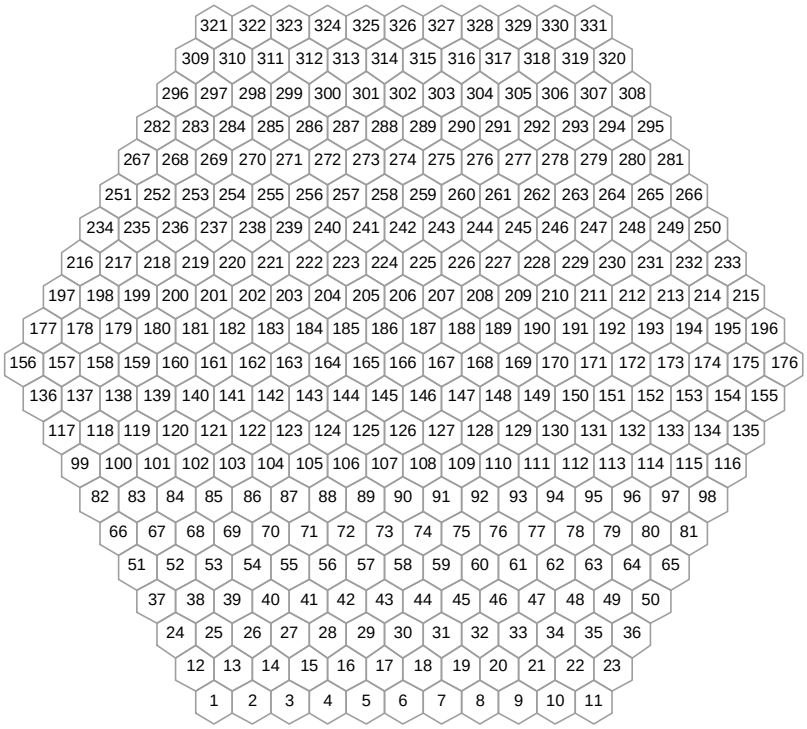

Figure 22. Indices of a Hexagon Frame

For each cell, the row and column increments to get the 6 neighbor cells in hexagon lattice start at the right and go around counter-clockwise. The details are listed below:

Row increments – even rows are shifted 1/2 cell to the right of odd rows

```
 1   1
0  *  0
-1  -1
```

Column increments - the column increment to get neighbors depends on the row's even/odd status.

For even row:

```
  0   1
-1  *  1
  0   1
```

For odd row:

```
  -1   0
 -1  *  1
  -1   0
```

A "neighbor matrix" is created so that for each cell, the cell numbers of its six neighbors can be retrieved. The dissimilarity matrix is then created for the cases. I studied simple cluster examples such as five clusters, with sizes 10, 20, 30, 40 and 50 respectively. My study of cluster layouts simplifies the dissimilarity matrix. The dissimilarity of two cases in the same cluster is 0; the dissimilarity of cases in different clusters is a constant that can be made arbitrarily large. The dissimilarity matrix is augmented by a row and a column for the generic empty cases, and by another row and column for the generic border cases. This requires four parameters: occupied cell to border cell cost, empty cell to border cell cost, occupied cell to empty cell cost, and empty cell to empty cell cost.

The cost associated with the layout is implicitly the sum of distances to the cases in the six neighbors of the cells, adding over all the cases in the grid. In practice the algorithm keeps track of the cost reduction when exchanging cases. The algorithm has a parameter giving the maximum allowed cycles and a stopping parameter that indicates how many cycles to have without a cost reduction before exiting. During each cycle, the

algorithm randomly creates pairs of the cells without replacement and drops one cell if the number of cells is odd. The two cases in each pair of cells have a chance to be swapped, thus at most one randomly selected cell is not considered in one cycle when there are odd number of cells. This is different from the preliminary algorithm where all the pairs were randomly picked. It might take a long time for a cell to get picked.

Graphics provide insights into how the algorithms work. Figure 23 shows the initial state for one test. The five colors represent the five clusters in the example dataset created. The numbers below the cell number in the occupied cells are the case numbers. The four parameters are indicated at the bottom of the Figure.

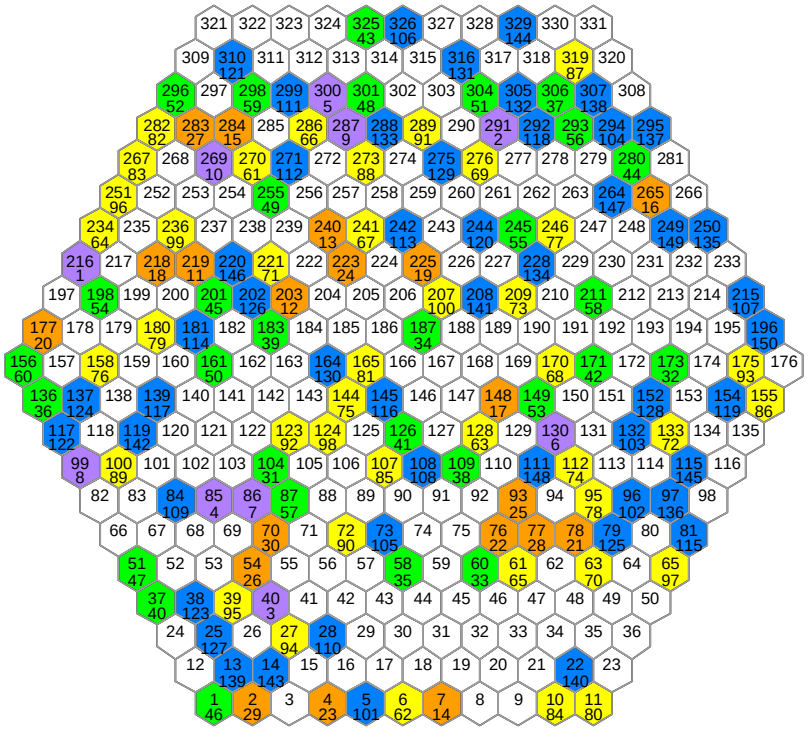

Figure 23. Initial State of the Cases

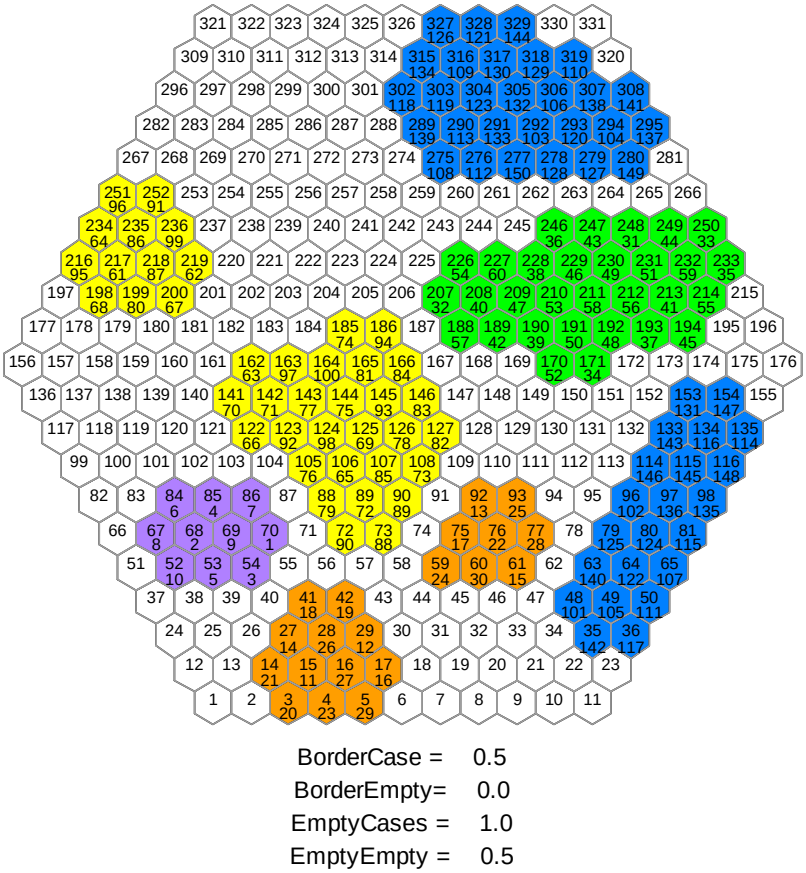

Figure 24. Clusters of Cases after 2000 Cycles

Figure 24 shows the view when the algorithm stops. The cost for cases in different clusters being adjacent is high enough so that no clusters touch. This example provides an insight that was previously missed. The insight comes from the yellow, orange and blue clusters being split into two pieces. There is clearly plenty of white space for the yellow clusters to join. This means that the cost function leads to stable states that are less than ideal. Previously the assumption was that partial clusters could be trapped

from growth because the within-grid bordering white cells were only one cell thick and were bordered by cells from another cluster. This can be viewed as getting stuck in a local minimum. The insight motivates thinking about solutions.

One possible solution to avoid the stable states includes costs that consider two rings of cells around each grid cell. A second solution is to keep track of each local cluster size and break ties by moving cases of the smaller sub-cluster to the larger sub-cluster. The algorithm's bookkeeping gets more and more complex with no guarantee of a good solution. Hence my attention turns to directly laying out the cases in clusters in a hexagon grid with a 'better start' instead of a random start, and then refining positions of cases within each cluster. I address the refinement step now as it uses basically the same algorithm that was just discussed, and address the new hexagon cluster graph layout in Section 4.3.

The cases within each cluster can be exchanged so that similar cases are closer together. This is the situation of exchanges without white space. Figure 25 is an initial placement of a cluster in a hexagon grid. The values for cases 1, 3 and 15 are then altered to make them outliers with respect to their own cluster.

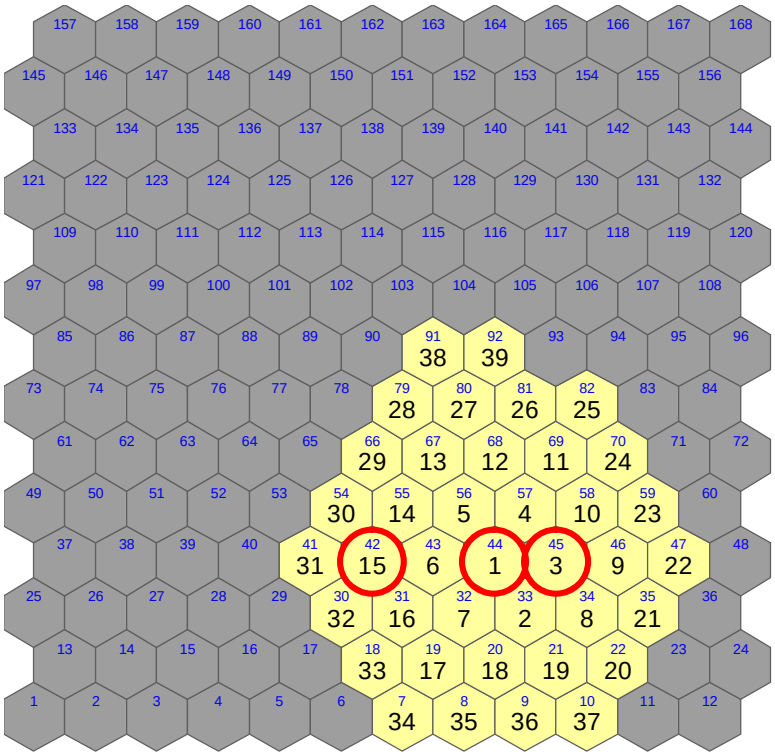

Figure 25. Case Placement without Sorting

Figure 26 is the placement after applying the exchange algorithm. As can be seen, cases 1, 3, and 15 are at the outside edge of the cluster. The neighboring empty cells around the outside act like ‘outlier fly paper’. Outlier cases don’t pay a distance cost for being by the neighbor cells that are empty. They have a bigger cost if they are surrounded by six occupied cells. Outliers in a cluster may jump to some place else in the outer ring of a cluster, but will not return to the interior unless competing against a more extreme outlier also located in the interior.

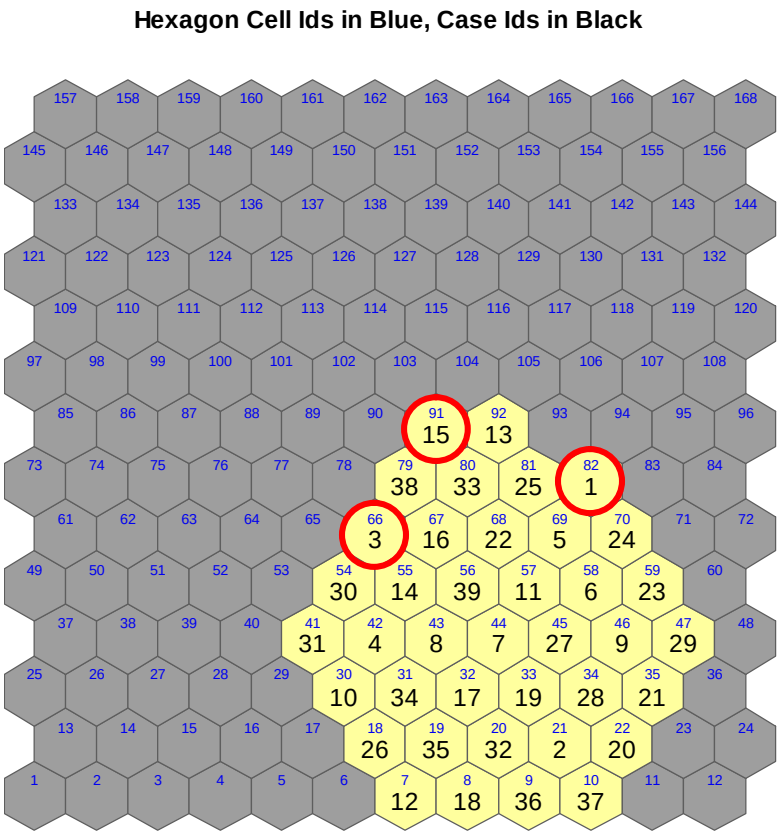

Figure 26. Case Placement after Sorting

4.3 Hexagon Cluster Layout Algorithm (HCL)

The Hexagon Cluster Layout (HCL) algorithm begins by explicitly laying out clusters chosen for display in a hexagon grid. It then uses the HOE algorithm in section 4.2 to refine the placement of cases within each cluster, so more cases are placed by similar cases based on a cost function.

Several steps are involved in the explicit layout of cases of clusters. The first step of the general procedure is to obtain a mean vector or representative vector for each

cluster. If data is available, the cases of each cluster can be averaged. If the average is in higher dimensions, classical multidimensional scaling can be used to obtain 2-D locations that attempt to preserve the higher dimensional distances between pairs of means. Then circles with areas proportional to cluster sizes are associated to the corresponding locations. A Lennard-Jones spring model with a central force adapted from a paper by Sun, Smith and Caudell (2003) then moves the circles to be tight about the center, but at least one hexagon cell apart. The resulting circle centers are overlaid on a hexagon grid to produce the seed cells of the layout. The hexagon layout algorithm assigns cases of each cluster to its seed cell and to neighboring cells in expanding hexagon rings about the seed cell.

I illustrate the layout of cluster summaries using clusters of 5^o x 5^o grid cells of the earth. The underlying data is similar to the AIRS Level 3 data described in chapter 3. There the layout was for multiple multivariate multi-altitude cluster summary vectors for 36 5^o x 5^o grid cells. For this example there are 2,387 grid cells of the possible 36 x 72 = 2,592 grid cells. This is because NASA's data processing algorithms can not adequately process some of the data such as that obtained over high altitude mountain ice. This, for example, excludes grid cells over Antarctica.

NASA continues to refine its processing algorithms, and the summaries in this chapter are based on a refinement that was not used for the summaries shown in chapter 3. Even this example is not based on the very latest algorithms. The focus here is on the layout methodology and not on detailed data interpretation. The Level 3 summaries used here are adequate for this purpose.

The clustering of earth grid cells here uses agglomerating clustering based on a 2,387 x 2,387 expected distance matrix provided by Amy Braverman from the Jet Propulsion Laboratory. The expected distance calculation is different from common distance calculation with one vector for each case being compared. Here each grid cell has multiple summary vectors. The calculation involves bivariate probability weighted sum of distances between suitably scaled summary vectors of all pairs of vectors involving one vector from each grid cell. Details are available in Braverman and Fetzer (2005).

The decision to use 20 clusters in the example is based on the need to restrict the number of colors to show the grid cell cluster membership of the whole earth. Twenty colors may seem too many, but there are eight 'singleton' clusters (cluster of size 1) and black could be used for all of them. The twenty clusters appear in Figures 27 and 28.

In this complex situation, applying classical multidimensional scaling to the expected distance matrix yields a 2-D "mean" vector for each grid cell. These are averaged to obtain a mean for each cluster of grid cells. The spring model moves these around substantially. The preservation of distances among the pseudo cluster means is not crucial. The most important thing is that the clusters don't overlap.

Figure 27 shows how the spring model lays out the 20 clusters as circular disks. The circle areas are proportional to the number of grid cells in the clusters.

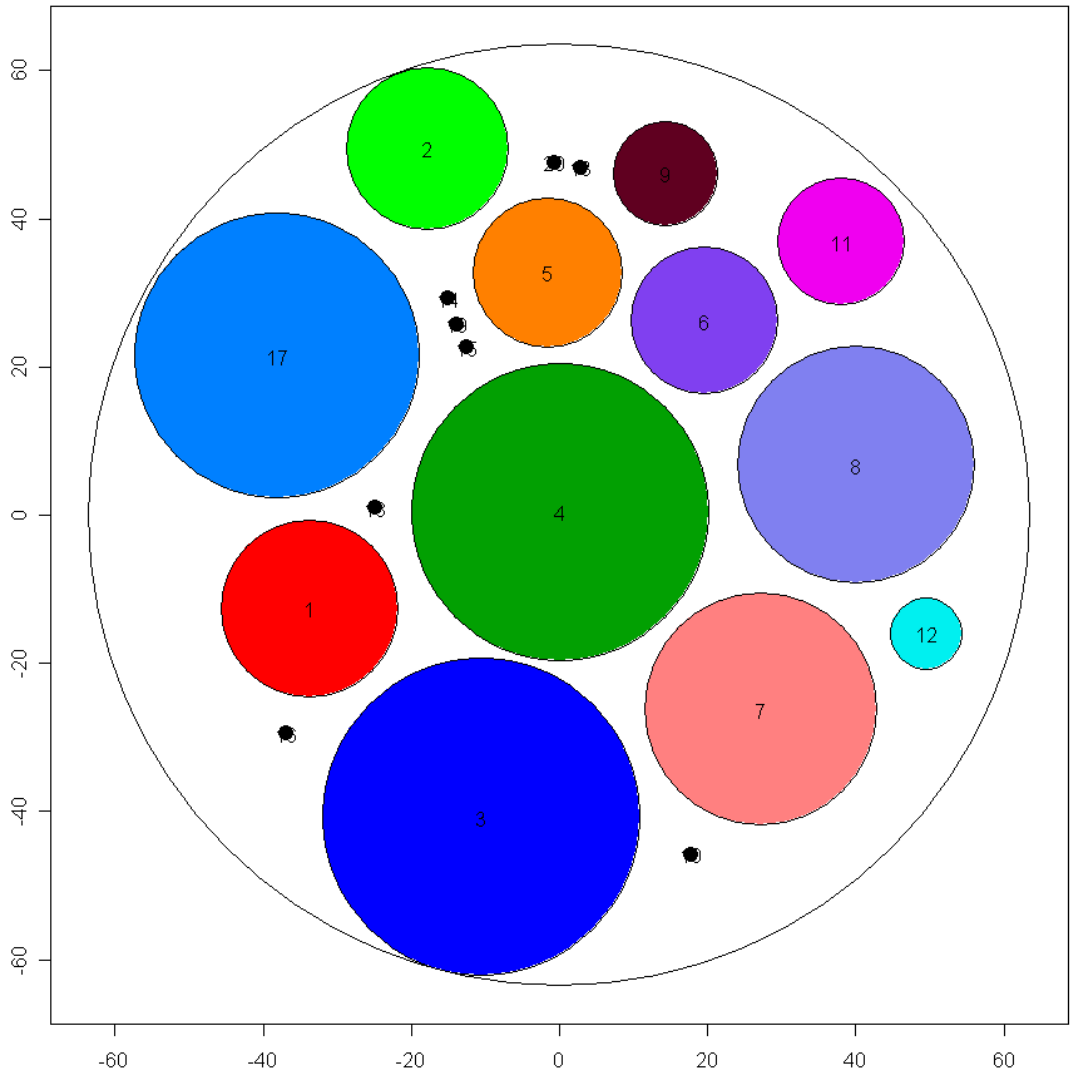

Figure 27. Spring Model Results to Determine Cluster Centroids

Figure 28 shows the cells on a hexagon grid. The sizes and positions of the clusters correspond to the spring model layout shown in Figure 27. The advantage of placing them in a hexagon grid is that there is a clear definition of neighbors and the

HOE algorithm further modifies the layout within each cluster, so that similar grid cells are usually positioned together.

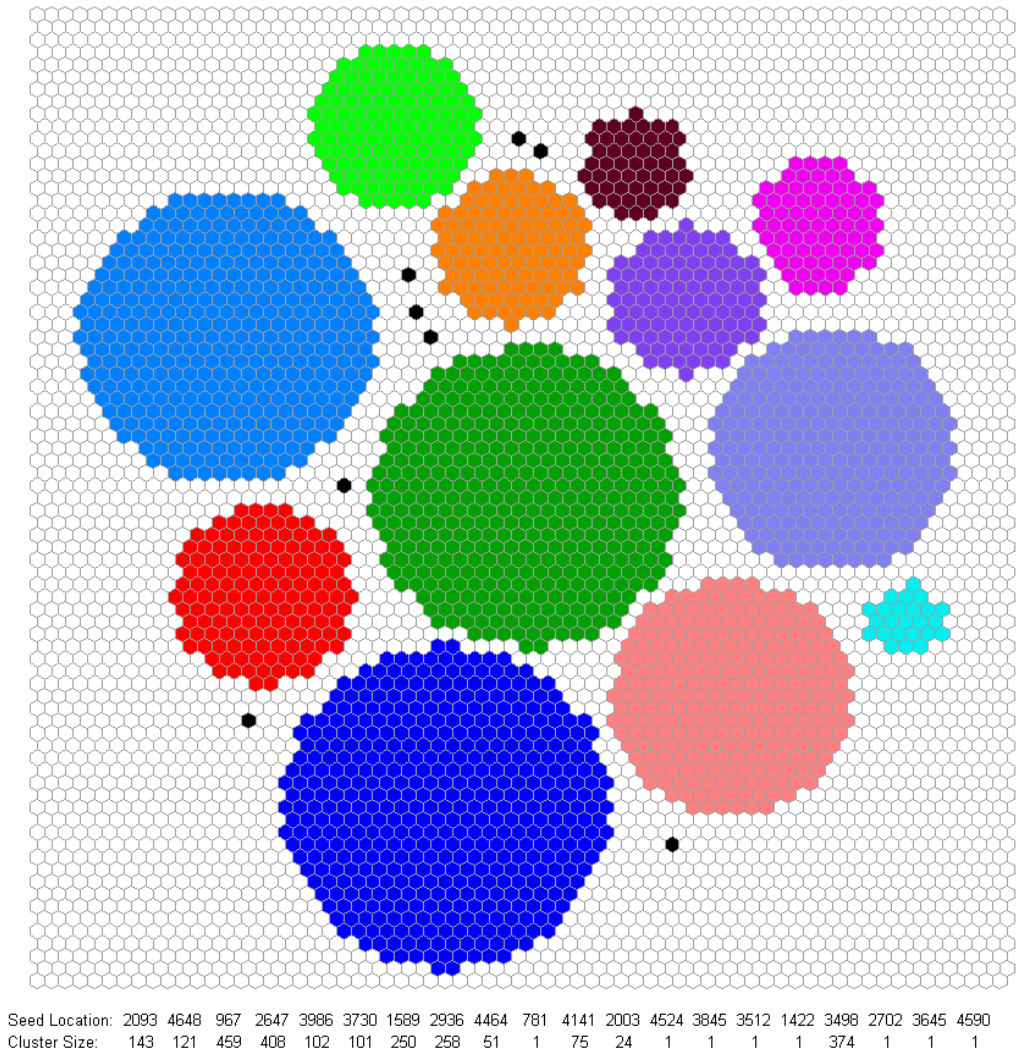

Figure 28. A Hexagon Grid Layout for the Clustered Global Grid Cells

The ‘Cluster Size’ row at the bottom of the plot indicates the number of cases within each cluster. The ‘Seed Location’ row displays the location of each corresponding

cluster's center. The location is given by the hexagon index in the grid. The indices of the hexagons in the grid were assigned using the same method described previously and shown in Figure 22. For example, the dark blue cluster at the bottom of the plot has 459 cases in it, and its center is placed at the 967^{th} hexagon cell.

Carr maps the clusters to a world map and shows one multivariate atmospheric cluster on a second level for the winter 2002 data, as shown in Figure 29. This represents the hexagon cluster cells as spheres in a plane and the grid cells of the earth as spheres in a parallel plane. The design connects the spheres representing the same grid cell on the earth with rendered tubes. The software called 'Glisten' (Carr 2004) provides sliders used to filter out all the hexagon cluster cells and tubes except for one cluster shown in gray. It is interesting to notice the atmospheric similarity during the north hemisphere winter between the Sahara desert and the southern U.S.

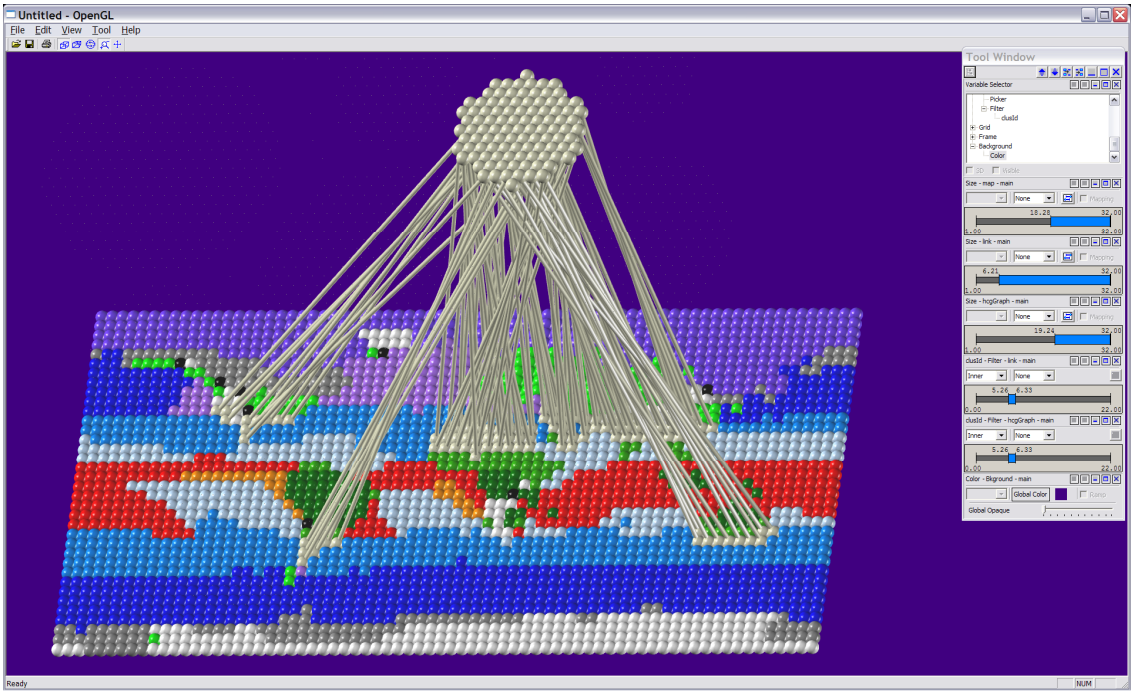

Figure 29. Parts of the Same Multivariate Atmospheric Cluster – Winter 2002
Source: Carr and Braverman (2007)

4.4 Summary

In this chapter I introduced two algorithms to layout cases in a hexagon grid in 2-D. The Hexagon Occupant Exchange algorithm and Hexagon Cluster Layout algorithm can be used together to produce Hexagon Cluster Graphs (HCG). HCG uses include:

- Calling attention to clusters,
- Showing sub-clusters inside clusters,
- Showing distances to cases in neighbor cells after first organizing cases to reduce the distances, e.g. showing a distance gap in lines between neighboring nodes,
- Displaying node coordinates for linked brushing,
- Displaying edge coordinates for linked brushing,

- Displaying glyphs, icons or pictures at nodes, and
- Dynamic graphics framework that includes panning and zooming, progressive disclosure, queries, linked brushing and other options..

Chapter 5. Round Polytope Layout in 3-D

Card et al. (1999) observes that "Although the challenges associated with 3-D have made it less desirable to many people, there are also good reasons to use 3-D to visualize information. Perhaps the most obvious advantage is the additional dimension to encode information. However, a subtler but profound advantage is that this additional dimension projects from the viewpoint toward infinity, creating a large visible workspace for holding visualizations and the results of information work. With the advent of mass-market graphically agile computers, 3-D visualization is likely to become more common". Recently, Ware and Mitchell (2008) undertakes a graph comprehension study using a very high resolution stereoscopic display with points and lines shown as lighting model rendered spheres and tubes. The results show an order of magnitude increase in the size of useful 3-D graphs over 2-D graphs. Therefore, I extend the 2-D CAP algorithm to 3-D, and discuss the extension of the HOE and HCL algorithms to 3-D in chapter 6 on future work.

Slicing convex polytopes with planes is the generalization of slicing convex polygons with lines. I choose to focus on the truncated octahedron in 3-D, because like the hexagon in 2-D, it is the roundest polytope that partitions the space. Truncated

octahedron is one of the 13 Archimedean solids. It has 14 faces, 36 edges, and 24 vertices. Figure 30 shows a truncated octahedron.

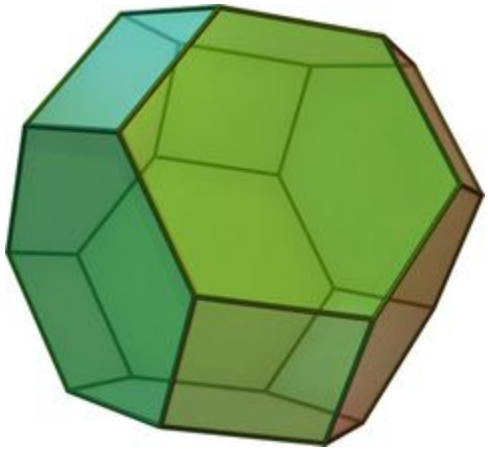

Figure 30. A Truncated Octahedron
Source: Wikipedia

A truncated octahedron has six square faces and eight hexagon faces. Assume it is centered at the origin. Let the orientation and size be such that the centers of the square faces are on the X, Y, Z axis respectively, and their coordinates are (0,0,1), (0,0,-1), (0,1,0), (0,-1,0), (1,0,0), (-1,0,0). The diagonals into the eight octants of the coordinate system intersect the eight hexagon faces at: (0.5 0.5 0.5), (-0.5 0.5 0.5), (0.5 -0.5 0.5), (-0.5 -0.5 0.5), (0.5 0.5 -0.5), (-0.5 0.5 -0.5), (0.5 -0.5 -0.5), (-0.5 -0.5 -0.5).

There are seven lines that run through the center of the truncated octahedron to the centers of paired faces. This corresponds to the three lines that are perpendicular to the hexagon sides in 2-D. There are twelve lines through the origin to each pair of

vertices. This corresponds to the three lines to the opposite vertices in the hexagon. Cutting from these 'vertex' directions provides the possibility of cutting off vertices to accommodate outliers. These 7+12=19 directions are used to partition a truncated octahedron.

In the cluster layout context, the task of the generalized 2-D CAP algorithm is to recursively partition a convex polytope with a plane, such that the volumes of the two sub-polytopes are proportional to the sizes of the two sub-clusters, and the two sub-polytopes are as round as possible. Thus two major tasks are involved, one is convex polytope volume calculation, and the other is the definition and calculation of a 'roundness' measure. Conway and Sloane (1982) discuss calculation of volumes and second moments of polytopes, and give the formula of the dimensionless second moment in *d* dimensions as

$$G_d = \frac{1}{d} \frac{\int_P \| x - \underset{\sim}{x} \|^2 \, dx}{(\int_P dx)^{(d+2)/d}}$$

This dimensionless second moment can be used as a measure of roundness. Table 1 lists the roundness of various shapes in 2-D and 3-D.

Table 1. Roundness for Various Shapes

	Square	Hexagon	Circle	Cube	Truncated Octahedron	Sphere
Roundness	0.08333	0.0802	0.0796	0.08333	0.07854	0.07697

5.1 Truncated Octahedron Slicing - Lattice Point Based Approximation

To approximate the volume and dimensionless second moment of polytopes, I generate a high resolution body-centered cube lattice and clip away points outside the truncated octahedron. The body-centered cube lattice is composed of two sets of cube lattices, one of which is shifted so that each point is at the center of a cube from the other lattice (except for the edges). Thus lattice point generation is easy.

The idea is that the truncated octahedron and each sub-polytope are partitioned based on how many cases are in each branch of the hierarchical structure. The truncated octahedron is cut along each direction mentioned above, and the direction that produces polytopes of points with the smallest roundness is chosen. The dimensionless second moment of a polytope is approximated by the sum of squares of points about the point average, divided by the number of points in the polytope, and adjusted for the volume of the polytope. I calculate the initial volume-per-point constant and use this constant times the number of points in each sub-polytope to approximate the volume of an irregular polytope.

A chosen body-centered cube lattice for a unit cube consisted of 250,000 points is clipped back to a truncated octahedron with 118,585 points. The dimensionless second moment calculated for the truncated octahedron is 0.08027; when starting with 2 million lattice points, the result truncated octahedron has 973,899 points and the approximated roundness is 0.07939. Theoretically the dimensionless second moment for truncated

octahedron is 0.07854 (Conway and Sloane 1982). The error is about 1% and the dimensionless second moment seems to be converging to the theoretical value as the number of lattice points grows. Thus the lattice based calculation is a reasonable approximation.

During each partitioning, there are almost always two possible cuts along each direction to produce the correct volume proportion. I pick the direction that produces the 'roundest' pair of sub-polytops based on the smallest sum of dimensionless second moments. There could be ties on the smallest roundness. For now I just pick the direction of the first occurrence. A possible alternative is to pick randomly. The partitioning goes on until every leaf node in the hierarchical structure gets its own polytope.

Graphics can help to show what the algorithm is doing. Various 3-D software packages are available to display 3-D graphs, including Glisten and RGL. The advantage of using RGL is that the results can be shown directly after running the R program for recursive partitioning, rather than exporting data and then importing it into another software package.

Figure 31 from Glisten shows the polytope after one cut when the targeted volume proportion is 0.3. The algorithm picks one direction that is perpendicular to a square face. When starting with 2 million lattice points the roundness of the resulting two sub-polytopes is 0.08858 and 0.1049 respectively.

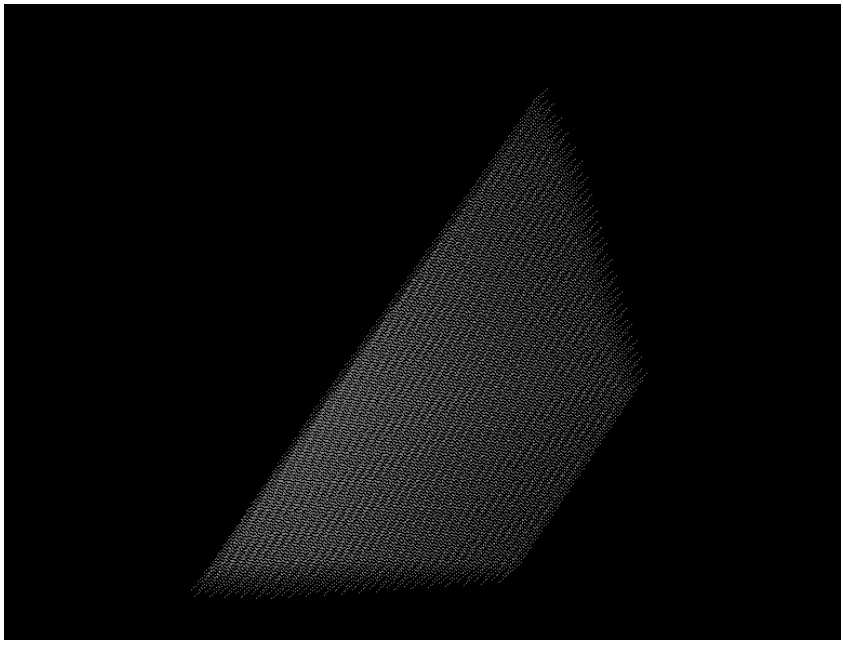

Figure 31. First Cut of a Truncated Octahedron Shown in Glisten

Figure 32 shows a resulting sub-polytope in RGL and the targeted volume proportion is 0.005. The algorithm cuts off a vertex of the truncated octahedron instead of using a plane parallel to a face of the truncated octahedron as in the proportion of 0.3 case, shown in Figure 31. This agrees with the intuition and the purpose of the algorithm. Starting with 2 million lattice points the roundness of the resulting two sub-polytopes is 0.07877 and 0.1889 respectively.

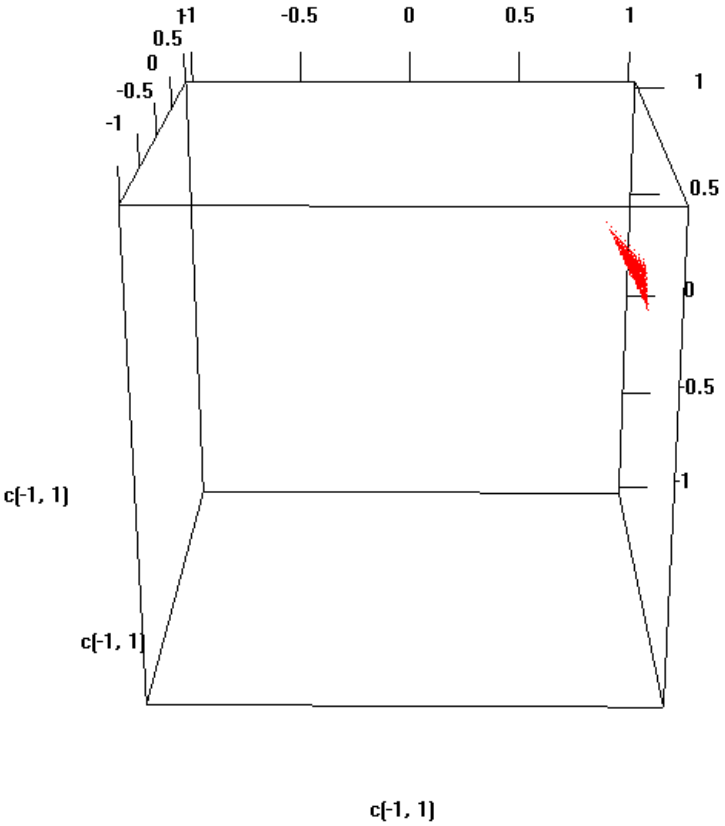

Figure 32. First Cut of a Truncated Octahedron Shown in RGL

I applied the algorithm to the 382 cases from the AIRS data described in Section 3.3. Figure 33 shows the cases at the center of the resulting individual polytopes from the recursive partitioning.

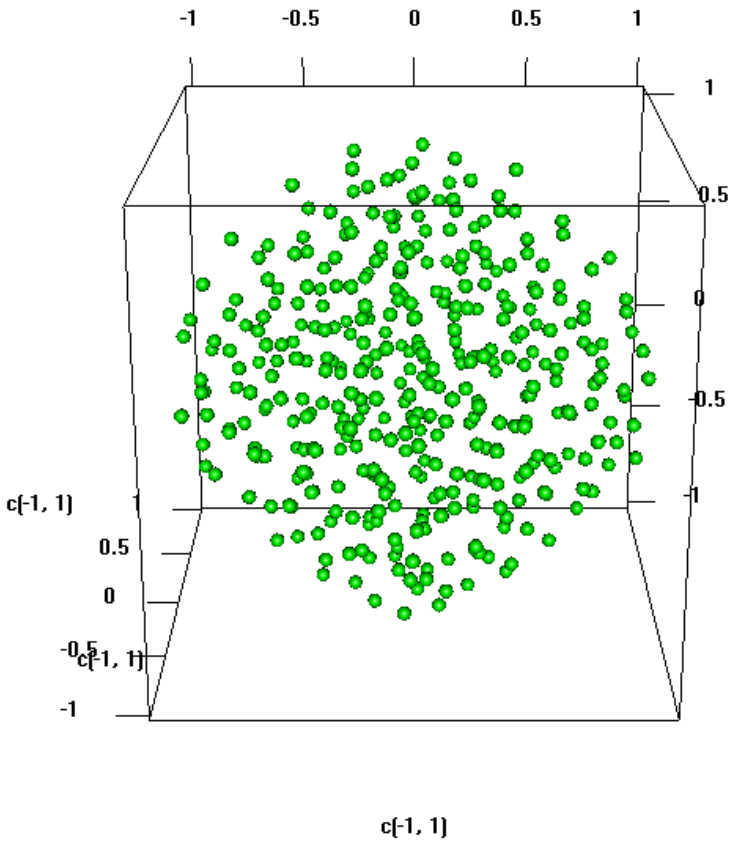

Figure 33. AIRS Data Layout in 3D

The 3-D locations of the cases and the scaled data are loaded into Glisten to be visualized. A glyph is plotted for each case. To simplify the example's appearance, only the 11 temperature variables are included in the signed deviation star glyphs. Pairs of consecutive variables are colored red, green, blue, yellow, and cyan. Altitude 11 is colored magenta. Figure 34 shows the initial position of the cases.

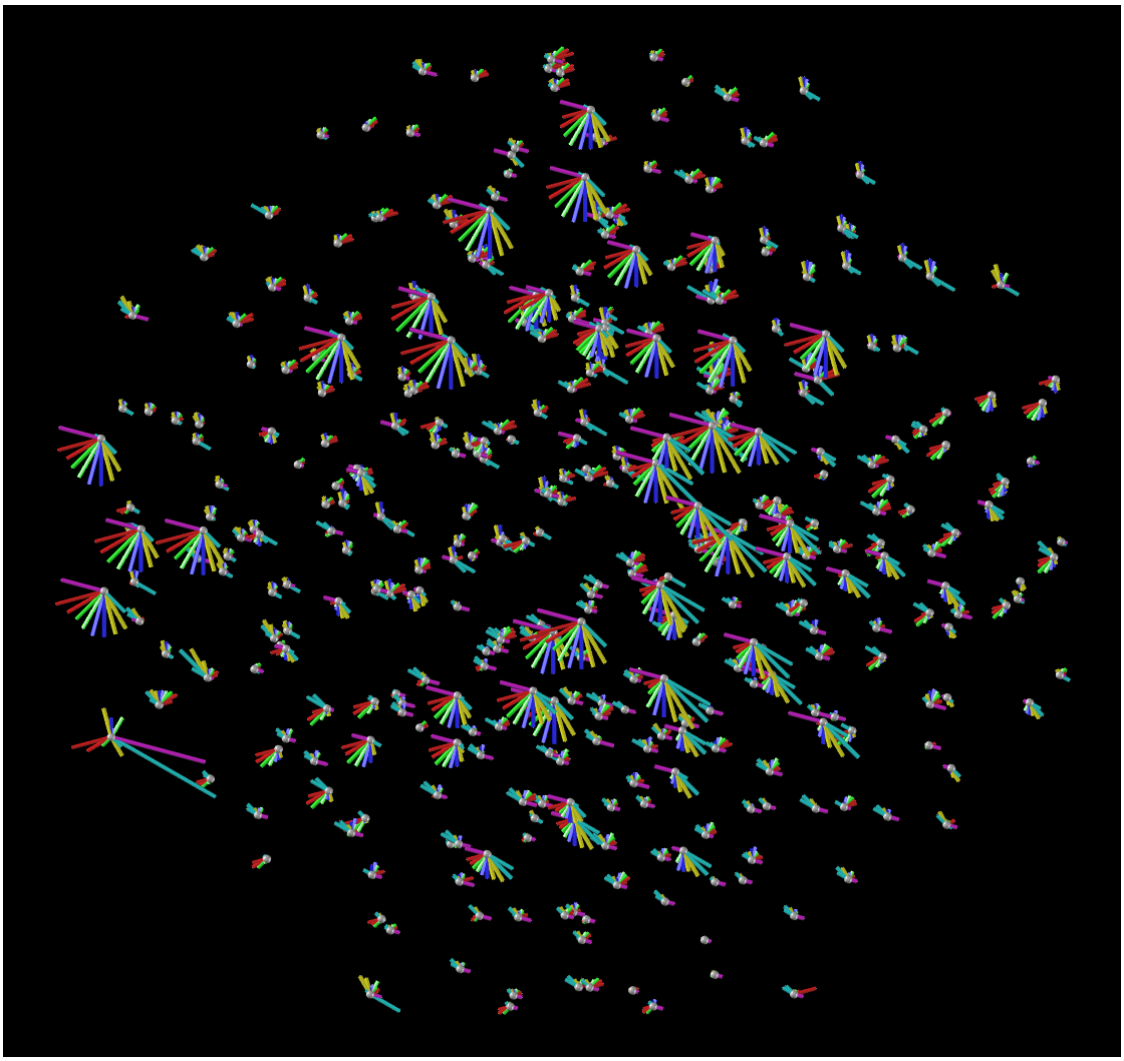

Figure 34. AIRS Data Layout in Glisten with Glyphs

The case near the left edge and towards the bottom is the outlier located at the bottom of Figure 16. As shown, there are many small values for the temperature variables represented as long rays. Figure 35 shows a rotated view of Figure 34. Looking from this angle, we can see the similar cases together on the right side, which corresponds to the cases on the right side in Figure 16.

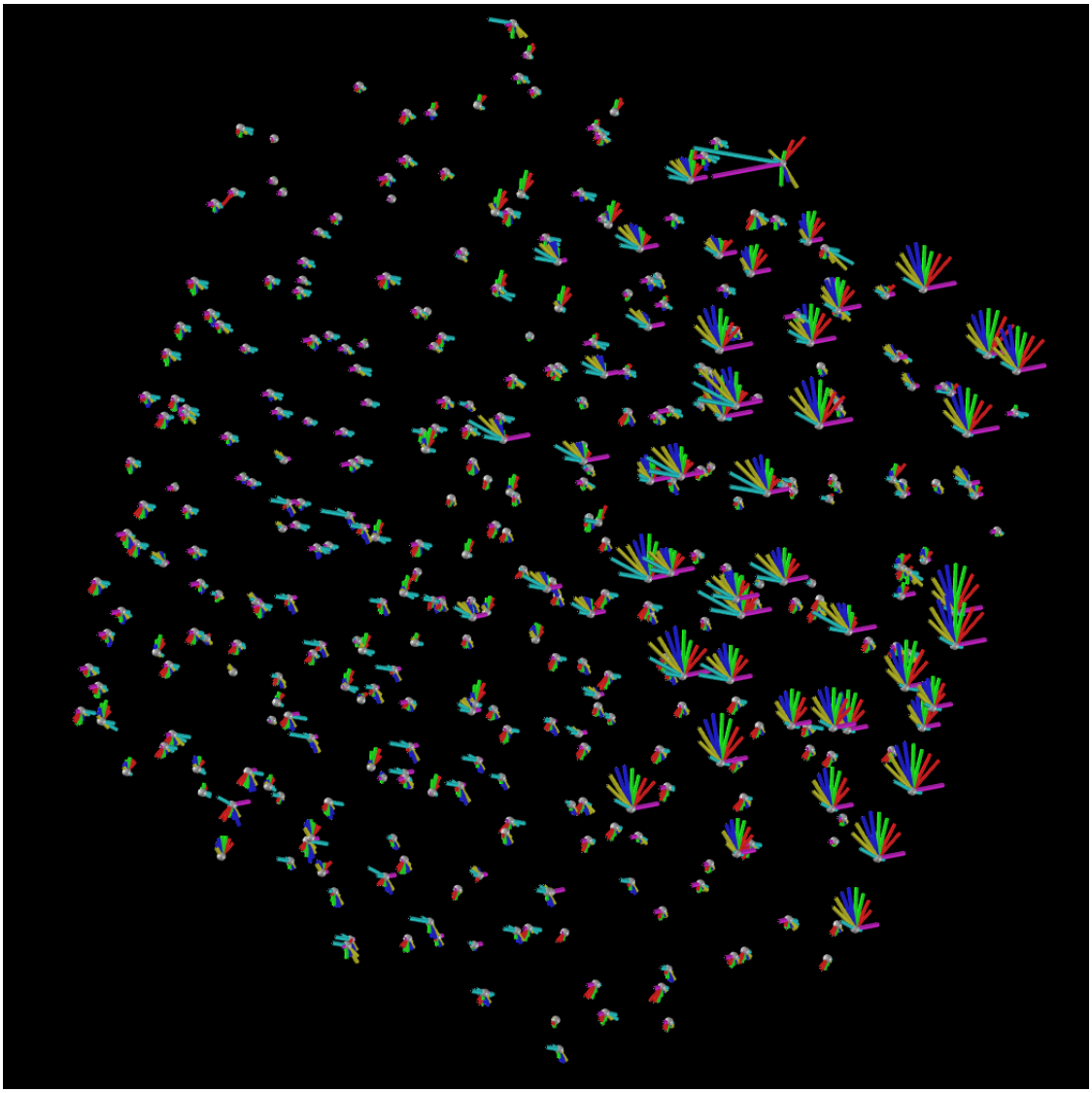

Figure 35. AIRS Data Layout in Glisten with Glyphs – A Different View

Besides rotation, Glisten provides powerful tools such as filtering. We can view and investigate one or two variables at a time and filter out all other variables. Figure 36 shows only temperature variables at altitude levels 3 and 4.

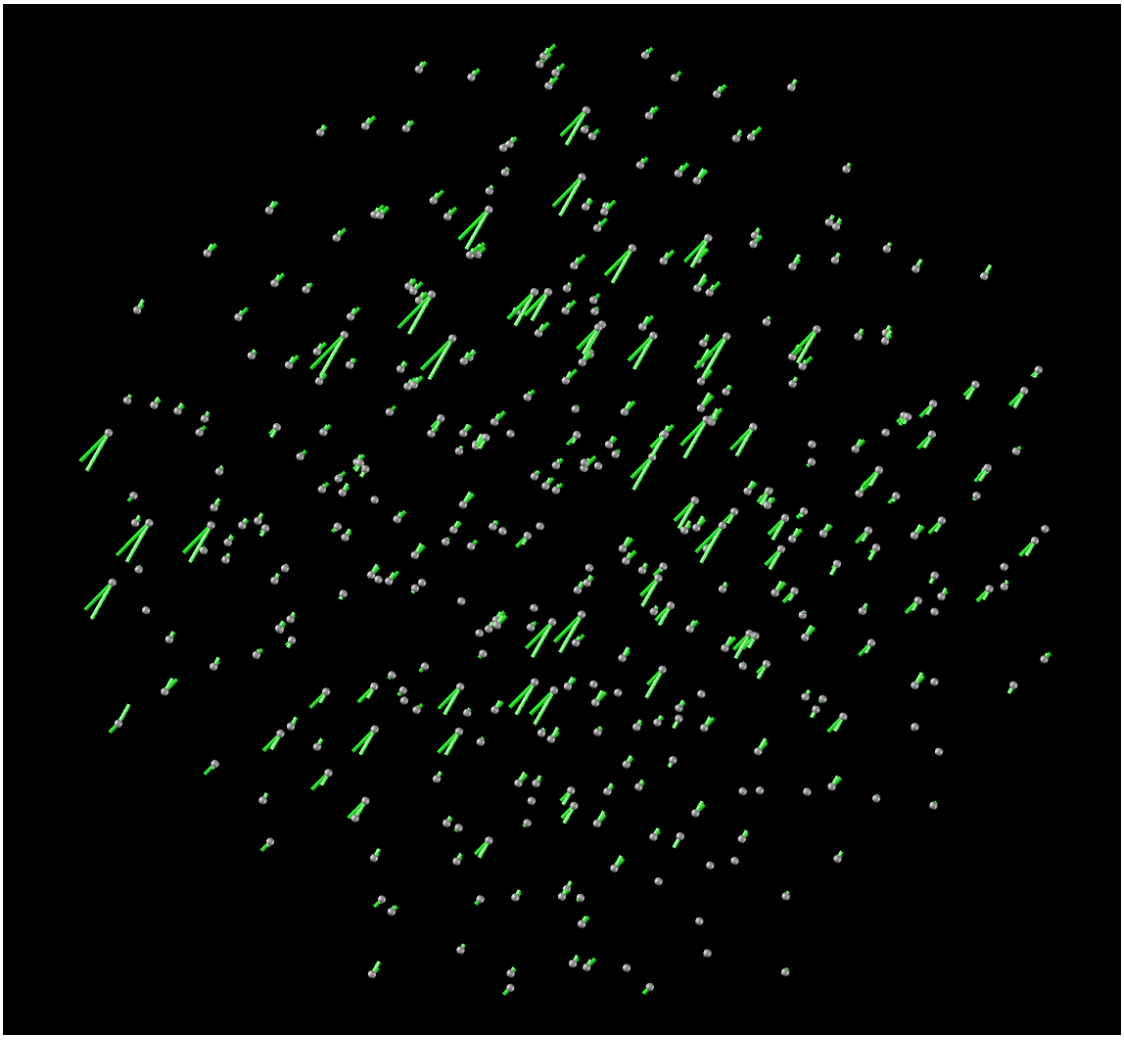

Figure 36. AIRS Data Layout in Glisten with Glyphs – Selected Variables

The filtered view simplifies the appearance and helps readers concentrate on fewer variables. The case locations represented by gray spheres are more visible. The rotation that helps to convey depth is not shown. The lighting model for tubes and spheres helps with depth perception but it is not necessarily well rendered in print. Larger glyphs would also improve the perception. It is desirable for the glyphs to always stay in a plane during rotation. Carr et at. (1988). Glisten needs to be adapted to do this.

The distribution of roundness and number of points for the individual polytopes doesn't change much by increasing the number of initial lattice points from 250,000 to 2 million. Table 2 shows the comparisons. In both cases, there was at most one point difference in terms of the number of points in the individual polytopes. The minimum, median and maximum roundness of the polytopes for the 2 million-points case are slightly smaller, and the mean is about the same. The R script's running time for the 250,000-points case is about 1 minute, while for the 2 million-points case it is about 10 minutes on a Dell Latitude D620 laptop, with Intel Duo T2400 @ 1.83GHz CPU and 2GB RAM.

Table 2. Comparison Between 250,000 and 2 million Initial Lattice Points

# Initial Lattice Points	**250,000**	**2,000,000**
# Points in Truncated Octahedron	118,585	973,899
Minimum # Points in Polytopes	310	2,549
Maximum # Points in Polytopes	311	2,550
Minimum Roundness	0.0879	0.0865
Median Roundness	0.1025	0.1022
Mean Roundness	0.1056	0.1056
Maximum Roundness	0.1901	0.1869

The truncated octahedron resulted from the 250,000 initial lattice points works pretty well. Speed can be improved by recoding in C or other compiled language.

5.2 Truncated Octahedron Slicing - Solving Geometric Equation Approach

The solving geometric equations approach involves calculating exact polytope volumes and dimensionless second moments rather than using lattice point approximations. These results could be used to evaluate the lattice based approximation approach in terms of speed and accuracy.

There are previous studies and results regarding convex polytope volume and dimensionless second moment calculation, including Conway and Sloane (1982), Lawrence (1991), etc. Online tools like CGAL (Computational Geometry Algorithms Library), Qhull, Polymake, etc. might also be helpful. The following result for prismatoid volume calculation can be used to address the value proportion partitioning task.

A prismatoid is a polyhedron where all vertices lie in two parallel planes (If both planes have the same number of vertices, it is called a prismoid.). If the areas of the two parallel faces are A_1 and A_3, the cross-sectional area of the intersection of the prismatoid with a plane midway between the two parallel faces is A_2, and the height (the distance between the two parallel faces) is h, then the volume of the prismatoid is given by $V = h(A_1 + 4A_2 + A_3)/6$. This formula can be used to calculate the volumes of the two sub-polytopes as a plane sweeps through the polytope.

Given the areas at the ends and middle of a prismatoid, the volume for each prismatoid can be computed. The areas of the end planes can be calculated directly from the vertices involved. Cutting the edges of the prismatoid half way between the end

planes gives the intersection points to compute the area of the middle plane, and the area completes what is needed to compute the volume of the prismatoid.

Consider the problem of finding the cutting plane location for the prismatoid that produces the targeted volume proportion. Assuming there is a quadratic change in the areas cross section when moving from one end of the prismatoid to the other, I can solve for the three coefficients of this quadratic equation based on the areas *A1*, *A2* and *A3*. The integral of a quadratic equation is a cubic equation that gives the volume as a function of the distance from one end of the prismatoid. I can then solve this cubic equation to get the candidate two locations for a cutting plane as needed.

Finding the centroids of the resulting prismatoids and getting the dimensionless second moment about the centroids seems harder. It is possible to feed the polytope's vertices to a symbolic software to get the needed integrals. Another approach could use results from Conway and Sloane (1982). One result for *n*-dimensional simplexes makes it possible to find the second moment of any figure, provided that it can be decomposed into simplexes.

Theorem: Let P be an arbitrary simplex in R^n with vertices $v_i=(v_{i1}, \ldots v_{in})$ for $0 \le i \le n$, then

a) the centroid of P is at the barycenter $\bar{v} = (v_0+\ldots+v_n)/(n+1)$ of the vertices;

b) $vol(P) = \frac{1}{n!}\det\begin{vmatrix} 1 & v_{01} & \ldots & v_{0n} \\ 1 & v_{11} & \ldots & v_{1n} \\ & & \ldots & \\ 1 & v_{n1} & \ldots & v_{nn} \end{vmatrix}$; and

c) the normalized second moment about the origin 0 is

$$I_0 = \frac{n+1}{n+2}\left\|\bar{v}\right\|^2 + \frac{1}{(n+1)(n+2)}\sum_{i=0}^{n}\left\|v_i\right\|^2$$

There are algorithms for decomposing a convex polytope into simplexes that can support the use of this result. This whole calculation approach is feasible but not as simple as it may seem.

5.3 Summary

In this chapter I extend the CAP algorithm to 3-D partitioning of convex polytopes and investigate a truncated octahedron as a starting point. The lattice point based approximation approach works pretty well in terms of resulting polytopes' uniform number of points and roundness. I also discuss the current research status and the difficulties of using the solving geometric equation approach to address the truncated octahedron slicing problem.

Chapter 6. Conclusions and Future Work

This dissertation develops 2-D and 3-D layouts that are useful to facilitate the visual study of multivariate data. The layout algorithms compute locations for plotting cases so that similar cases are close together. The locations can be used for linking the cases to other views of the data. The classic example is linked brushing to views such as scatterplot matrices. One example shows 3-D rendered tubes linking cases between two parallel plane views of the cases. The tubes highlight specific clusters of the data. The layouts typically emphasize the display of cluster structure.

The layouts algorithms I developed emphasize producing locations that are surrounded by round polygons in 2-D or round polytopes in 3-D. Roundness is defined as the dimensionless second moment about the polygon or polytope centroid. This supports the display of round multivariate glyphs. The algorithms fall into two classes that are applicable in either two or three dimensions. The first class recursively slices polygons or polytopes with lines or planes respectively to produce pairs of polygons or polytopes that will hold two sub-clusters of a binary cluster tree. The centers of the polygons or polytopes become the plotting locations for representing the leaf nodes of the cluster tree. The algorithms are successful in addressing the tasks of linking and plotting case

descriptors such as multivariate glyphs. The regular polygon layouts can be used to organize images.

I produced the 2-D slicing algorithm called CAP in 2001. However, an example was not published until 2005 when a paper addressing the same task appeared using a competing algorithm based on iterative Voronoi tessellations. The slicing algorithm allows specification of the orientation of the cutting lines. The example uses six orientations, so the points-surrounding polygons at various levels of clustering appear simple. The Voronoi produced polygons have edges whose orientations are not restricted, so the plots appear busier from one perspective, but tessellations have their own elegance. For many situations the choice between the two algorithms likely makes little practical difference.

In terms of 3-D slicing, the locations produced by my algorithm are useful for plotting glyphs as demonstrated. The R scripting language provides adequate speed for modest applications with a few hundred cases. The algorithm will scale to much larger examples when written to produce a compiled version. The polytope roundness distribution suggests a little room for improvement. The roundness average is somewhat better than the roundness of a tetrahedron. Future work will study a modification of the criterion for picking the best orientation from the slicing plane options for each cut. I will study different weighted averages of the roundness values for the two polytopes produced by slicing.

The 3-D slicing implementation uses an approximation based on a high resolution grid of points. My work in developing equation solving and integration implementation is

not complete. I intend to finish this work and see how close the approximations are to the equation results. Currently I only have a roundness comparison result for a truncated octahedron, and the 1% error can be decreased by use of a finer grid.

I am not aware of an iterative 3-D Voronoi region implementation. If this has not been done I may pursue it.

The second layout approach places cases on regular lattice points. The near neighbor regions for the 2-D algorithms developed here are hexagons, the roundest regular polygons that tessellate 2-D. I developed new HOE algorithms that include the use of empty cells to call attention to different clusters.

The HOE algorithm still has a problem in putting cases of the same cluster together. Future research will pursue exchanging larger block of cells and/or modifying the cost function to avoid the stable states that prevent parts of clusters from joining.

The HCL algorithm solves the problem of keeping the cases of the same cluster together. Future research will extend this to 3-D layouts with truncated octahedron cells. The generalization of the spring models, the placement of cases in a grid and the exchanging of cases within each cluster in 3-D are straightforward. The bookkeeping is a little more complicated, since each truncated octahedron has 14 neighbors (six neighbors share square faces, and eight neighbors share hexagon faces) instead of 6 neighbors for hexagons.

I plan to produce two R packages, one for the recursive partitioning and the other for the hexagon cluster layout. The current algorithms are implemented primarily as R scripts. Recoding the key portions in C will improve the speed. Some alternative

implementations may become available for parts of the computations. For example, partitioning a polytope by planes is an active area in computer graphics. The new tools and methods developed in other fields may help in statistical visualization applications.

This research also develops graphics to show additional information. I enhanced Wills' (1998) recursive partitioning algorithm for cases of clusters by adding white space to visually separate clusters at each level of partitioning. I showed the HOE algorithm would push the outliers of the cluster to the boundary of the layout when used to reorganize cases within a cluster. This is helpful in assessing clusters.

I developed two new star glyphs called signed deviation and folded deviation star glyphs. These provide a more symmetric representation of small and large values than traditional glyphs. The cost involves making it harder to see variation in intermediate values. The strong benefit is being able to see variation in small values.

I will also continue the research on 2-D and 3-D layouts of multivariate objects. This includes continuing development of dissimilarity measures for complicated objects such as models, molecules, dendrograms would be appropriate. I will also dedicate efforts in further improving visualization tools by adding more features. Each class of objects viewed is likely to have its own type of graphics and manipulation needs. For example, display and manipulation methods can be quite different for genes, molecules, cluster trees, models and earth grids. I am also interested in doing research in layouts and graphs in parallel planes with Daniel Carr and Michael Trosset.

The layouts on a 2-D surface could be extended to a sphere, especially for the earth data and its visualization/presentation. The advantage of a sphere is that

conceptually all things are there even though not all of them can be viewed at the same time. Google developed tools that allow users to map the Google maps to a sphere as a texture mapping. Similar techniques might be applied.

Finally I can pursue layouts in 4-D. 4-D views can be linked 2-D views, an animated 3-D view, or simple 3-D glyph views. Of course the increased preservation of inter-point distances trades off against the difficulty humans have judging the distance in these representations.

```
######################### CAP Algorithm in SPLUS ##############################
inter = function(l1,l2){
       tmp = l1-l2
       x = -tmp[1]/tmp[2]
       y = (l2[1]*l1[2]-l1[1]*l2[2])/tmp[2]
       return(x,y)
}

#_____________________________________________________________________________
allvertices <- function(lmx){
    nc <- ncol(lmx)
   nr <- 2*nc*(nc-1)
   vpmx = matrix(0,nrow=nr ,ncol=2)
   subs = 1:4-4
   for(i in 1:(nc-1)){
      for(j in (i+1):nc){
       s1 <- slopes[i]
      s2 <- slopes[j]
      subs = subs+4
      tmp1 = c(lmx[1,i],s1)
      tmp2 = c(lmx[2,i],s1)
      tmp3 = c(lmx[1,j],s2)
      tmp4 = c(lmx[2,j],s2)

      vp1 <- inter(tmp1,tmp3)
      vp2 <- inter(tmp1,tmp4)
      vp3 <- inter(tmp2,tmp3)
      vp4 <- inter(tmp2,tmp4)
      vpmx[subs,] <- rbind(vp1,vp2,vp3,vp4)
      }
   }
   vpmx <- matrix(unlist(vpmx),ncol=2)
   vpp <- rep(1,nr)
   eps <- .00001

   for (i in 1:nr){
   itp <- vpmx[i,2]-slopes*vpmx[i,1]    # y0-b*x0
   for(j in 1:ncol(lmx)) {
   rg <- range(lmx[1:2,j])
   if((itp[j]<(rg[1]-eps)) || (itp[j]>(rg[2]+eps))) vpp[i]<-0
   }
}
if(sum(vpp)>0) fvp <- matrix(vpmx[vpp==1],ncol=2)
return(fvp)

}

#_____________________________________________________________________________
polyarea <- function(vertex){
  n <- nrow(vertex)
  subs <- c(2:n,1)
  x1 <- vertex[,1]
  x2 <- vertex[subs,1]
  y1 <- vertex[,2]
  y2 <- vertex[subs,2]
  area <- sum((x2-x1)*(y1+y2))/2
  return(abs(area))
```

```
}

#_____________________________________________________________________________
plotarea <- function(fvp){
hull <- chull(fvp)
plot(fvp[,1],fvp[,2],type='n',xlab='',ylab='',axes=F)
polygon(fvp[hull,1],fvp[hull,2],density=0,col=3)
}
#_____________________________________________________________________________
howround <- function(vertex){

center <- apply(vertex, 2, mean)
s1 <- (vertex[,1]-center[1])^2
s2 <- (vertex[,2]-center[2])^2
s <- sum(s1+s2)/nrow(vertex)^2
s <- s/polyarea(vertex)
return(s)
}
#_____________________________________________________________________________
resultpolys <- function(v1,v2,icpt,slp){

vv <- rbind(v1,v2)
xx <- vv[,1]
od <- order(xx)
x <- xx[od]

tmp <- c(T,diff(x)>.000001)
od <- od[tmp&c(tmp[-1],T)]
vv <- vv[od,]
if(nrow(vv)<3) vv <- rbind(vv,v1[abs(v1[,2]-slp*v1[,1]-icpt)<.005,])
hull <- chull(vv)
return(vv[hull,])

}
#_____________________________________________________________________________
ptstolns <- function(vertex){

n <- nrow(vertex)
rstlines <- matrix(0,nrow=2,ncol=6)
for (i in 1:6){
  diff <- vertex[,2]-slopes[i]*vertex[,1]
  ord <- order(diff)
  rstlines[1,i] <- diff[ord[1]]
  rstlines[2,i] <- diff[ord[n]]
}

 return(rstlines)
}
#_____________________________________________________________________________
nodup <- function(v){

xx <- 1000*v[,1]+v[,2]
od <- order(xx)
x <- xx[od]
if(sum(diff(x)<.00001)>0)
{
plc <- seq(along=x)[c(F,diff(x)<.00001)]
od <- od[plc]
return(v[-od,])
} else
```

```
return(v)
}

#_____________________________________________________________________________
splitit <- function(fraction,ab){

vert <- allvertices(ab)
vert <- nodup(vert)
hull <- chull(vert)
orivert <- vert[hull,]
totarea <- polyarea(orivert)

rdness <- 1000

for(i in 1:6)
{
  st <- ab[1,i]
  ed <- ab[2,i]
  kk <- 1

  while((ed-st)>0.01 || kk==1)
  {
  kk <- 0
  ab1 <- ab
  ab1[1,i] <- (st+ed)/2
  fvp1 <- allvertices(ab1)
  fvp1 <- nodup(fvp1)
  hull <- chull(fvp1)
  area <- polyarea(fvp1[hull,])
  myline <- area - fraction*totarea
  if(myline<0) {ed <- (st+ed)/2} else {
                st <- (st+ed)/2  }
  }

  ppoly <- resultpolys(orivert, fvp1[hull,],(st+ed)/2,slopes[i])
  thisrd <- howround(ppoly)+howround(fvp1[hull,])

  if( thisrd < rdness)
  {
   rdness <- thisrd
   poly1 <- fvp1[hull,]
   poly2 <- ppoly
  }
}

  hull <- chull(poly2)
  poly2 <- poly2[hull,]

return(rdness,poly1,poly2)

}

#_____________________________________________________________________________
ang <- (c(0,60,120)+15)*pi/180
ang1 <-(c(150,90,30)+15)*pi/180
angs <- c(ang,ang1)

slopes <- tan(angs)

a <- rep(2,3)/cos(ang)
```

```
ab <- rbind(-abs(a),abs(a))
fvp <- allvertices(ab)
hull <- chull(fvp[,1],fvp[,2])
fvp <- fvp[hull,]

graphsheet()
par(oma=c(0,0,0,0),mar=c(0,0,0,0),pin=c(8,8))
plotarea(fvp)
ab <- ptstolns(fvp)

#_____________________________________________________________________
clus.size <- function(mat){
   n <- ncol(mat)
   vec <- rep(0,n)
   for( i in 1:n){
      j <- mat[1,i]
      k <- mat[2,i]

      if(k < 0) vec[i] <-   2 else
      if(j < 0) vec[i] <-   1+ vec[k] else vec[i] <- vec[j]+vec[k]
   }
   return(vec)
}

clus <- data.clust

mat <- t(clus$merge)
dis <- clus$height
n  <- ncol(mat)
nsize <- clus.size(mat)
drawpoly <- fvp
polys <- matrix(0,nrow=n-1,ncol=12)
polys <- rbind(polys,as.vector(ab))
p.point <- matrix(0,nrow=2,ncol=n+1)

y <- n
for (i in 1:2){
   node <- y[1]
   y <- y[-1]
   leaf <- mat[,node]
   cutit <- polys[node,]
   cutit <- matrix(cutit, nrow=2)
   j <- leaf[1]
   k <- leaf[2]
   if(j < 0) jsize <- 1 else jsize <- nsize[j]
   if(k < 0) ksize <- 1 else ksize <- nsize[k]
   fraction <- jsize/(jsize+ksize)

   res1 <- splitit(fraction, cutit)
   res2 <- splitit(1-fraction, cutit)

   if(res1$rdness < res2$rdness)
    {
      poly1 <- res1$poly1
      poly2 <- res1$poly2
    } else
    {
      poly1 <- res2$poly2
      poly2 <- res2$poly1
    }
```

```
    area1 <- ptstolns(poly1)
    area2 <- ptstolns(poly2)

   if(jsize==1){
      j <- abs(j)
      p.point[,j] <- apply(poly1,2,mean)
   } else
       polys[j,] <- as.vector(area1)

   if(ksize==1){
      k <- abs(k)
      p.point[,k] <- apply(poly2,2,mean)
   } else
        polys[k,] <- as.vector(area2)
   if (dis[n-i+1]>0.0) {
   drawpoly <- rbind(drawpoly, c(NA,NA), poly1, c(NA,NA), poly2)
   }
   y <- c(y,leaf[leaf>0])
}

graphsheet()
par(oma=c(0,0,0,0),mar=c(0,0,0,0),pin=c(8,8))

plotarea(fvp)
polygon(drawpoly,density=0,col=8, lwd=1)

ppp <- p.point[p.point!=0]
ppp <- matrix(ppp,ncol=2,byrow=T)
points(ppp,cex=.6,col=1,pch=16)
```

```
############################## HOE in R ########################################
psubs = 1:3    # penalty subscripts
sameClusCost = 0
diffClusCost = 3
borderCaseD =  .5   # the border to case distance
borderEmptyD = 0   # the border to empty distance
emptyCaseD =   1   # the empty to case distance smaller than between clusters
emptyEmptyD = .5   # the empty to empty distance

# 1.  Increments matrix for accessing cell neighbors

rInc = c(0,1,1,0,-1,-1)

cInc = matrix(c(
1,1,
0,1,
-1,0,
-1,-1,
-1,0,
0,1),ncol=2,byrow=T)

# 2. Indexing the allowed locations within a hexagon matrix

hSize = 10
cellMatrix = matrix(0,nrow=2*hSize+1,ncol=2*hSize+1)

cellMatSize=2*hSize+1
   k=0
   for (i in -hSize:hSize){
           L=cellMatSize-abs(i)
           jL=trunc((-L)/2)
        jU=trunc((L-1)/2)
   for (j in jL:jU){
      k=k+1
      cellMatrix[i+hSize+1,j+hSize+1]=k
   }}

#3. Create a neighbor matrix indexed by allowed cell

nValid = max(cellMatrix)
neib = matrix(0,nrow=6,ncol=nValid)

   k=0
   for (i in -hSize:hSize){
           iL=cellMatSize-abs(i)
           jL=trunc((-L)/2)
        ju=trunc((L-1)/2)
           ind = abs(i)%%2+1
         for(j in jL:ju){
            k=k+1
            ni=i+rInc
            nj=j+cInc[,ind]
               good = abs(ni)<=hSize & abs(nj)<=hSize
               if(any(good))
   neib[(1:6)[good],k]=cellMatrix[cbind(ni+hSize+1,nj+hSize+1)[good,]]
   }}
neib[neib==0] = nValid+1   # border cells

#4. Create distance matrix for clusters of cases
```

```
clustSize= c(10,20,30,40,50)
nCase =sum(clustSize)
clustCum =cumsum(clustSize)

caseDistMat = matrix(diffClusCost,ncol=nCase,nrow=nCase) # distance

for (i in 1:length(clustSize)){
   sizeN = clustSize[i]
   je=clustCum[i]
   jb= je-sizeN+1
   ind=jb:je
   subs = as.matrix(expand.grid(x=ind,y=ind))
   caseDistMat[subs] = matrix(sameClusCost,nrow=sizeN,ncol=sizeN)
}

diag(caseDistMat)=0 # set distance from case to self as 0

vec = rep(emptyCaseD,nCase)
caseDistMat = rbind(caseDistMat,vec)
caseDistMat = cbind(caseDistMat,c(vec,emptyEmptyD))

vec = rep(borderCaseD,nCase+1)
caseDistMat = rbind(caseDistMat,vec)
caseDistMat = cbind(caseDistMat,c(vec,borderCaseD))
subs= matrix(c(nCase+1,nCase+2,nCase+2,nCase+1),ncol=2)
caseDistMat[subs] = borderEmptyD

#5. Create cell member vector for allowed locations

emptyCase = nCase+1
borderCase = nCase+2

cellMember = rep(emptyCase,nValid+1)
casePosition = sample(1:nValid,nCase,replace=F) #where to place cases
cellMember[casePosition]=1:nCase                # placing them
cellMember[nValid+1]=borderCase                 # universal border cell

cellMemberOld= cellMember    # save first state for comparison

#6. Create cost vector for allowed locations

cost = rep(0,nValid)
for (i in 1:nValid){
   locs= neib[,i]
#   cost[i] = sum(caseDistMat[cellMember[i],cellMember[locs]])
  cost[i] = sum(sort(caseDistMat[cellMember[i],cellMember[locs]])[psubs])
}

costBegin = sum(cost)

#7. Graphics feedback setup

nr = nrow(cellMatrix)
nc = ncol(cellMatrix)
x=1:nc
y=(1:nr)*sqrt(3)/2
cents = expand.grid(x=x,y=y)
cents$x = cents$x + c(rep(rep(c(0,.5),c(21,21)),10),rep(0,21))
```

```
good = t(cellMatrix)>0
cellX=cents$x[good]
cellY=cents$y[good]
cellId = t(cellMatrix)[good]
polyX  = c(-.5,0,.5,.5,0,-.5,NA)
polyY  =  c(-1,-2,-1,1,2,1,NA)/(2*sqrt(3))
centsReps = rep(7,length(cellX))

# 8. Looping constants and initial values

valids = 1:nValid
nMatPairs = nValid%/%2

# 9.  Pick cell pairs and swap if closer to neighbors

costReduce = 0
for (m in 1:2000){
   sam = matrix(sample(valids,2*nMatPairs,replace=F),nrow=2)
for(k in 1:nMatPairs){
   locs = sam[,k]                    # pair of locations
   midCases = cellMember[locs]       # candidate cases to swap

   if(min(midCases)==emptyCase)next  # skip if both cells are empty
   a=locs[1]
   b=locs[2]
   A=midCases[1]
   B=midCases[2]

   neibsA = neib[,a]
   neibsB = neib[,b]
   neibCasesA = cellMember[neibsA]
   neibCasesB = cellMember[neibsB]
#   costCur=sum(caseDistMat[A,neibCasesA])+
#          sum(caseDistMat[B,neibCasesB])
   costCur=sum(sort(caseDistMat[A,neibCasesA])[psubs])+
           sum(sort(caseDistMat[B,neibCasesB])[psubs])

   #
   if(any(neibsA==b)){
      neibCasesA[neibsA==b] = A
      neibCasesB[neibsB==a] = B
   }


#   costsBA = c(sum(caseDistMat[B,neibCasesA]),
#               sum(caseDistMat[A,neibCasesB]))
   costsBA = c(sum(sort(caseDistMat[B,neibCasesA])[psubs]),
              sum(sort(caseDistMat[A,neibCasesB])[psubs]))
   costNew = sum(costsBA)
   if(costCur> costNew){
       cost[locs] = costsBA
       cellMember[locs]= rev(midCases)
       costReduce = costReduce + costCur-costNew
   }
}
}

# 10.  Hexagon Plot

cellCase = cellMember[1:length(cellId)]
```

```
clust = cut(cellCase,breaks=c(0,10.5,30.5,60.5,100.5,150.5,nCase+2),labels=F)
mycolor = c("#B080FF","#FFA000","#00FF00","#FFFF00","#0080FF","#FFFFFF")

windows(width=8,height=8)
plot(c(0,22),c(0,22*sqrt(3)/2),
     type='n',axes=FALSE,ylab='',xlab='',
     main=paste('Last Cost Reduction =',round(costReduce)))
probsCh= format(rep(" ",4))
distCh = format(round(c(borderCaseD,borderEmptyD,emptyCaseD,emptyEmptyD),2))
mtext(side=1,line=-1,paste('BorderCase = ',probsCh[1],distCh[1]),cex=.8)
mtext(side=1,line=0, paste('BorderEmpty= ',probsCh[2],distCh[2]),cex=.8)
mtext(side=1,line=1, paste('EmptyCases = ',probsCh[3],distCh[3]),cex=.8)
mtext(side=1,line=2, paste('EmptyEmpty = ',probsCh[4],distCh[4]),cex=.8)
polygon(rep(cellX,centsReps)+polyX,
        rep(cellY,centsReps)+polyY,
        col=mycolor[clust])
polygon(rep(cellX,centsReps)+polyX,
        rep(cellY,centsReps)+polyY,
        density=0,col="#A0A0A0")
text(cellX,cellY+.2,cellId,cex=.55,adj=.5)
good = cellCase <= nCase
text(cellX[good],cellY[good]-.2,cellCase[good],cex=.55,adj=.5)
```

```
#####################Signed Deviation Star Glyph ########################

graphsheet(color.table=color.tb)
par(oma=c(0,0,0,0),mar=c(0,0,0,0),pin=c(8,8))

plotarea(fvp) # plot all
polygon(drawpoly,density=0,col=3, lwd=.5)

star.angs <- seq(0,180, length=35)[-c(1,35)]
star.angs <- star.angs*pi/180    # 40 degrees apart, 9 values to show

symbols(ppp[,1],ppp[,2], circles=rep(.3/9,nrow(ppp)), add=T, inches=F, col=3)
symbols(ppp[,1],ppp[,2], circles=rep(.6/9,nrow(ppp)), add=T, inches=F, col=3)

for(i in 1:nrow(mydata))
{
xx.b <- rep(ppp[i,1],33)
yy.b <- rep(ppp[i,2],33)
xx.e <- ppp[i,1]+.3*clus.data.s[i,]*cos(star.angs)/9
yy.e <- ppp[i,2]+.3*clus.data.s[i,]*sin(star.angs)/9

lines(c(ppp[i,1]-.6/9, ppp[i,1]+.6/9),c(ppp[i,2], ppp[i,2]),col=3)

segments(xx.b[1:11],yy.b[1:11],xx.e[1:11],yy.e[1:11],col=4, lwd=1)
segments(xx.b[12:22],yy.b[12:22],xx.e[12:22],yy.e[12:22],col=5, lwd=1)
segments(xx.b[23:31],yy.b[23:31],xx.e[23:31],yy.e[23:31],col=6, lwd=1)
segments(xx.b[32:33],yy.b[32:33],xx.e[32:33],yy.e[32:33],col=7, lwd=1)
}
```

```
################## Folded Deviation Star Glyph ##########################

graphsheet(color.table=color.tb)
par(oma=c(0,0,0,0),mar=c(0,0,0,0),pin=c(8,8))

plotarea(fvp) # plot all

symbols(ppp[,1],ppp[,2], circles=rep(.3/9,nrow(ppp)), add=T, inches=F, col=3)
symbols(ppp[,1],ppp[,2], circles=rep(.6/9,nrow(ppp)), add=T, inches=F, col=3)

for(i in 1:nrow(mydata))
{

xx.b <- rep(ppp[i,1],33)
yy.b <- rep(ppp[i,2],33)
xx.e <- ppp[i,1]+.3*abs(clus.data.s[i,])*cos(star.angs)/9
yy.e <- ppp[i,2]+.3*abs(clus.data.s[i,])*sin(star.angs)/9

pos <- clus.data.s[i,] > 0
neg <- clus.data.s[i,] < 0

sub1 <- (1:11)[pos[1:11]]
sub2 <- (1:11)[neg[1:11]]

segments(xx.b[sub1],yy.b[sub1],xx.e[sub1],yy.e[sub1],col=4, lwd=1)
segments(xx.b[sub2],yy.b[sub2],xx.e[sub2],yy.e[sub2],col=8, lwd=1)

sub1 <- (12:22)[pos[12:22]]
sub2 <- (12:22)[neg[12:22]]

segments(xx.b[sub1],yy.b[sub1],xx.e[sub1],yy.e[sub1],col=5, lwd=1)
segments(xx.b[sub2],yy.b[sub2],xx.e[sub2],yy.e[sub2],col=9, lwd=1)

sub1 <- (23:31)[pos[23:31]]
sub2 <- (23:31)[neg[23:31]]

segments(xx.b[sub1],yy.b[sub1],xx.e[sub1],yy.e[sub1],col=6, lwd=1)
segments(xx.b[sub2],yy.b[sub2],xx.e[sub2],yy.e[sub2],col=10, lwd=1)

sub1 <- (32:33)[pos[32:33]]
sub2 <- (32:33)[neg[32:33]]

segments(xx.b[sub1],yy.b[sub1],xx.e[sub1],yy.e[sub1],col=7, lwd=1)
segments(xx.b[sub2],yy.b[sub2],xx.e[sub2],yy.e[sub2],col=11, lwd=1)

}
```

```
#####################Simulation 3-D in R ##############################
## 1. Normal to cutting planes

dirc <- matrix(c(1,0,0,    ## Center of faces
                 0,1,0,
                 0,0,1,
                 1,1,1,
                 1,-1,1,
                 1,1,-1,
                -1,1,1,
                 1,.5,0,   ## Vertices
                 1,0,.5,
                 0,.5,1,
                 0,1,.5,
                .5,0,1,
                .5,1,0,
                 1,-.5,0,
                 1,0,-.5,
                 0,-.5,1,
                 0,1,-.5,
                 -.5,0,1,
                 -.5,1,0), ncol=3,byrow=TRUE)
dirc.len <- (dirc^2) %*% c(1,1,1)
dirc[,1] <- dirc[,1]/sqrt(dirc.len)
dirc[,2] <- dirc[,2]/sqrt(dirc.len)
dirc[,3] <- dirc[,3]/sqrt(dirc.len)
dirc = t(dirc)

## 2. Generate points in a bodycentered-cube-lattice bccube

ctrs <- as.matrix(expand.grid(c(.5,-.5),c(.5,-.5),c(.5,-.5)))  # the centers of
   8 hexagons

n <- 100
dat <- seq(-1,1,length=n)
cube <- as.matrix(expand.grid(dat,dat,dat))
dx <-(dat[2]-dat[1])/2
bccube <- rbind(cube,cube+dx)

# Remove points shifted outside the cube
bccube <- bccube[bccube[,1] <=1 & bccube[,2] <=1 & bccube[,3] <= 1,]
num.p <- nrow(bccube)  # point in the bccube

## 3. Remove points to get a truncated octahedron  TO

# Slice of pairs of opposite corners to create a truncated octahedron
TO <- bccube
for(i in 1:4){
   cur <- ctrs[i,]
   const <- sum(cur^2)
   d <- as.matrix(TO) %*% cur
   TO <- TO[abs(d)<=const,]
}

num.left <- nrow(TO)
num.left/num.p  # 0.49429  The target is .5
8*num.left/num.p # 3.95432: approximate truncated octahedron volume.

## 4. Thoughts and trial calculation of second dimensionless moment
```

```
mmt.whole <- sum(TO^2)/num.left
mmt.whole/(8* num.left/num.p)^(2/3)/3

## 5. Function Definition for comparing splits with the same fraction

con1= 3*(4/num.left)^(2/3)
con2=5/3
nd = ncol(dirc)

split.3d <- function(data, fraction){
   nr = nrow(data)
   nf = round(nr*fraction)
   GPS = matrix(0,nrow=nd,ncol=2)
   time.record <- rep(NA, nd)

   for (i in 1:nd){
      z1 = Sys.time()

      cc <- data %*% dirc[,i]
      ord = order(cc)
      subs=ord[1:nf]

      pointL <- data[subs,]
      pointH <- data[-subs,]

      gpL <- sum((scale(pointL,center=T,scale=F)^2))/(con1*nf^con2)
      gpH <- sum((scale(pointH,center=T,scale=F)^2))/(con1*(nr-nf)^con2)

      gp1 = gpL + gpH

      # cut from the opposite direction
      subs = ord[1:(nr-nf)]
      pointL <- data[subs,]
      pointH <- data[-subs,]

      gpL <- sum((scale(pointL,center=T,scale=F)^2))/(con1*(nr-nf)^con2)
      gpH <- sum((scale(pointH,center=T,scale=F)^2))/(con1*nf^con2)

      gp2 <- gpL + gpH

      GPS[i,]=c(gp1,gp2)
      z2 = Sys.time()
      time.record[i] <- difftime(z2, z1)
   }  #end for loop

   return(GPS)
}

fraction <- 0.3
GPS = split.3d(TO,fraction)

fraction = .0005
GPS = split.3d(TO,fraction)

## AIRS Example, ‘mat’ saves the cluster joining steps.

clus.size <- function(mat){
   n <- ncol(mat)
   vec <- rep(0,n)
```

```
   for( i in 1:n){
      j <- mat[1,i]
      k <- mat[2,i]

      if(k < 0) vec[i] <-   2 else
      if(j < 0) vec[i] <-   1+ vec[k] else vec[i] <- vec[j]+vec[k]
   }
   return(vec)
}

n  <- ncol(mat)     # number of joins
nsize <- clus.size(mat)

mylist=vector("list",n)
mylist[[n]] <- list(subs=1:nrow(TO))
p.point <- matrix(0, nrow=3, ncol=n+1)
round.mea <- matrix(0,nrow=n+1,ncol=2)

y <- n

for (i in 1:n){
  node <- y[1]
   y <- y[-1]
   leaf <- mat[,node]
   j <- leaf[1]
   k <- leaf[2]
   if(j < 0) jsize <- 1 else jsize <- nsize[j]
   if(k < 0) ksize <- 1 else ksize <- nsize[k]
   fraction <- jsize/(jsize+ksize)

   current.subs <- mylist[[node]]$subs
   dat <- TO[current.subs,]

   GPS <- split.3d(dat, fraction)

  temp <- min(GPS)
  colLeft = which(GPS[,1]==temp[1])
  colRight = which(GPS[,2]==temp[1])

  if(length(colLeft)>0){
    choseRow=colLeft[1]
    dir <- dirc[,choseRow]
    frac=fraction
    cc <- as.matrix(dat) %*% dir
    nf <- floor(nrow(dat)*frac)
    part <- order(cc)[1:nf]
    subs = current.subs[part]} else {

    choseRow=colRight[1]
    dir <- dirc[,choseRow]
    frac=1-fraction
    cc <- as.matrix(dat) %*% dir
    nf <- floor(nrow(dat)*frac)
    part <- order(cc)[(nf+1):nrow(dat)]
    subs = current.subs[part]}

  if(jsize==1) {
   j <- abs(j)
   p.point[,j] <- apply(dat[part,],2,mean)
   round.mea[j,1] <- nrow(dat[part,])
```

```
    round.mea[j,2] <-
    sum((scale(dat[part,],center=T,scale=F)^2))/(con1*(nrow(dat[part,]))^con2)
} else mylist[[j]] <- list(subs=subs)

  if(ksize==1) {
    k <- abs(k)
    p.point[,k] <- apply(dat[-part,],2,mean)
    round.mea[k,1] <- nrow(dat[-part,])
    round.mea[k,2] <-
sum((scale(dat[-part,],center=T,scale=F)^2))/(con1*(nrow(dat[-part,]))^con2)
 } else mylist[[k]] <- list(subs=current.subs[!is.element(current.subs,subs)])

  y <- c(y,leaf[leaf>0])
}

library(rgl)
open3d()
plot3d(c(-1,1),c(-1,1),c(-1,1),type='n')
plot3d(t(p.point),type='s',col='green',radius=.03,add=TRUE)
```

REFERENCES

REFERENCES

[1] Ankerst, M., S. Berchtold, and D. A. Keim (1998). "Similarity Clustering of Dimensions for an Enhanced Visualization of Multidimensional Data", *IEEE*. 52-60.

[2] Balzer, M. and O. Deussen (2005). "Voronoi Treemaps", *IEEE Symposium on Information Visualization.* 49-56. IEEE.

[3] Bartram, L. and C. Ware (2002). "Filtering and Brushing with Motion", *Information Visualization*, Vol.1, No.1, 66-79.

[4] Becker, R. A. and W. S. Cleveland (1987). "Brushing Scatterplots", *Technometrics*, 29, 127-142.

[5] Bederson, B. B., B. Shneiderman, and M. Wattenberg (2001). "Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies", *ACM Transactions on Graphics*, Vol.21, No.4, 833-854.

[6] Bertin, J. and W. J. Berg (1984). *Semiology of Graphics*. The University of Wisconsin Press.

[7] Borg, I. (1997). *Modern multidimensional scaling : theory and applications*. Springer, New York.

[8] Braverman, A. and E. Fetzer (2006). "A Probabilistic Approach to Mining Massive Earth Science Data Sets", *website http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/39338/1/05-3717.pdf.*

[9] Brewer, C. A. (1994). "Color Use Guidelines for Mapping and Visualization", *Visualization in Modern Cartography*, edited by A.M. MacEachren and D.R.F. Taylor, 123-147, Elsevier Science, Tarrytown, NY.

[10] Brower, J. C. and K. M. Kyle (1988). “Seriation of an original data matrix as applied to palaeoecology”, *Lethaia*, 21, 79-93.

[11] Bruls, M., K. Huizing, and J. J. van Wijk (2000). “Squarified treemaps”, *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, 33-42. Computer Society.

[12] Card, S. K., J. D. Mackinlay, and B. Shneiderman, Eds. (1999). *Readings in Information Visualization – Using Vision to Think.* Morgan Kaufmann.

[13] Carr, D. B. (1991). “Looking at Large Data Sets Using Binned Data Plots”, *Computing and Graphics in Statistics*, edited by A. Buja and P. Tukey, 7-39. Springer, New York.

[14] Carr, D. B. (1994a). “Converting Tables to Plots”, *Technical Report* No. 101, Center for Computational Statistics, George Mason University, Fairfax, VA, 22030.

[15] Carr, D. B. (1994b). “Using Gray in Plots”, *Statistical Computing & Graphics Newsletter*, Vol.5, No.2, 11-14.

[16] Carr, D. B. and A. Braverman (2007). “Visualizing Cluster-Compressed Multivariable and Multialtitude Atmospheric Data”, *Statistics Colloquium Series*, George Mason University. http://www.galaxy.gmu.edu/stats/colloquia/ColloquiaFall2007.html

[17] Carr, D. B., R. J. Littlefield, W. L. Nicholson, and J. S. Littlefield (1987). “Scatterplot Matrix Techniques for Large N”, *Journal of the American Statistical Association* Vol.82, No.398, 424-436.

[18] Carr, D. B. and W. L. Nicholson (1988). "EXPLOR4: A Program for Exploring Four-Dimensional Data." *Dynamic Graphics for Statistics*, edited by W. S. Cleveland and M. E. McGill, 309-329. Wadsworth, Belmont, California

[19] Carr, D. B. and A. R. Olsen (1996). “Simplifying Visual Appearance by Sorting: An Example Using 159 AVHRR Classes”, *Statistical Computing & Graphics Newsletter*, Vol.7, No.1, 10-16.

[20] Carr, D. B., A. R. Olsen, S. M. Pierson, and J. P. Courbois (2000). “Using Linked Micromap Plots to Characterize Omernik Ecoregions”, *Data Mining and Knowledge Discovery*, 4, 43-67.

[21] Carr, D. B. and M. H. Sung (2004). “Graphs for Representing Statistics Index by Nucleotide or Amino Acid Sequences,” Edited by J. Antoch. *Proceeding in Computational Statistics,* 73-83, Physica Verlag, New York.

[22] Carr, D. B., A. R. Olsen, and D. White (1992). “Hexagon Mosaic Maps for Display of Univariate and Bivariate Geographical Data”, *Cartography and Geographical Information Systems*, Vol.19, No.4, 228-236, 271.

[23] Carr, D. B. and R. Sun (1999). “Using Layering and Perceptual Grouping in Statistical Graphics”, *Statistical Computing & Graphics Newsletter*, Vol.10, No.1, 25-31.

[24] Chen, C. (2006). *Information Visualization*. Springer-Verlag London Limited.

[25] Chen, H. (2004). “Compound brushing explained”, *Information Visualization*, Vol.3, No.2, 96-108.

[26] Chernoff, H. (1973). “Using Faces to Represent Points in K-dimensional Space Graphically”, *Journal of the American Statistical Association*, Vol.68, 361-368.

[27] Cleveland, W. S. (1993). *Visualizing Data*. Hobart Press.

[28] Cleveland, W. S. (1994). *The Elements of Graphing Data*. Hobart Press.

[29] Cleveland, W. S. and R. McGill (1984). “Graphical Perception: Theory, Experimentation, and Application to the Development of Graphics Methods”, *Journal of the American Statistical Association* 79, 531-554.

[30] Conway, J. H. and N. J. A. Sloane (1982). ‘Voronoi Regions of Lattices, Second Moments of Polytopes, and Quantization’, IEEE transactions on information theory, Vol. IT-28, No.2.

[31] Conway, J. H. and N. J. A. Sloane (1999). *Sphere Packings, Lattices and Groups.* Springer Verlag.

[32] Davidson R. and D. Harel (1996). “Drawing Graphs Nicely Using Simulated Annealing”. *ACM Transactions on Graphics*, Vol.15, No.4, 301-331.

[33] Di Battista G. (1998). *Graph Drawing: Algorithms for the Visualization of Graphs*. Upper Saddle River, NJ: Prentice Hall.

[34] Eades, P. (1984). “A heuristic for graph drawing”, *Congressus Numerantium*, 42, 149-160.

[35] Eick, S. G. and G. J. Wills (1993). “Navigating Large Networks with Hierarchies”, *IEEE*. 204-210.

[36] Everitt, B. S. (1993). *Cluster Analysis, 3rd Edition*. Halsted Press New York.

[37] Forsell, C., S. Seipel, and M. Lind (2005). “Simple 3D Glyphs for Spatial Multivariate Data”, *IEEE Symposium on Information Visualization.* IEEE.

[38] Friedman, J. H. (1977). “A Recursive Partitioning Decision Rule for Nonparametric Classification”, *IEEE Transactions on Computers*, 404-408.

[39] Fruchterman T. M. J. and E. M. Reingold (1991). “Graph drawing by force-directed placement”, *Software-Practice and Experience*, 21, 1129-1164.

[40] Furnas, G. W. and A. Buja (1994). “Prosection Views: Dimensional Inference Through Sections and Projections”, Journal of Computational and Graphical Statistcis, Vol.3, No.4, 323-353.

[41] Gentle, J. E. (2005). *Elements of Computational Statistics*, Springer.

[42] Gnanadesikan, R. and R. K. Blashfield, et al. (Panel on Discriminant Analysis, Classification, and Clustering) (1989). “Discriminant Analysis and Clustering”, *Statistical Science*, Vol.4, No.1, 34-69.

[43] Greenacre, M. J. (1984). *Theory and Applications of Correspondence Analysis*. Academic Press.

[44] Grinstein, G. and H. Levkowitz, Eds. (1995). *Perceptual Issues in Visualization*. Springer, New York.

[45] Guo, D., D. Peuquet, and M. Gahegan (2002). “Opening the Black Box: Interactive Hierarchical Clustering for Multivariate Spatial Patterns”, *ACM, GIS ’02*.

[46] Hammer, O., D. A. T. Harper, and P. D. Ryan (2004). PAST-Palaeontological Statistics, Version 1.19 documentation. http://folk.uio.no/ohammer/past/index.html

[47] Hastie, T., R. Tibshirani, and J. H. Friedman (2003). *The Elements of Statistical Learning*, Springer.

[48] Hulle, M. M. V. (2000). *Faithful Representations and Topographic Maps*, Wiley-Interscience Publication.

[49] Johnson, B. J. and B. Shneiderman (1991). "Tree maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures", *Proceedings of the 2nd International IEEE Visualization Conference.* 284-291. IEEE Computer Society.

[50] Keller, P. (1993). *Visual Cues: Practical Data Visualization*. IEEE Computer Society Press.

[51] Kleiberg, E., H. van de Wetering, and J. J. van Wijk (2001). "Botanical Visualization of Huge Hierarchies", *Proceedings of IEEE Visualization Conference*.

[52] Kohonen, T. (1995). *Self-Organizing Maps*, Springer, Berlin.

[53] Koike, H. and H. Yoshihara (1993). "Fractal Approaches for Visualizing Huge Hierarchies", *Proceedings of IEEE Symposium on Visual Languages*. 55-60.

[54] Kosslyn, S. M. (1994). *Elements of Graph Design*. W.H.Freeman and Company, New York.

[55] Kosslyn, S. M. (2006). *Graph Design for the Eye and Mind*. Oxford University Press, New York.

[56] Krause, A. and M. Olson (2000). *The Basics of S and S-Plus, 2nd Edition*. Springer, New York.

[57] Kruskal, J. B. and M. Wish (1978). *Multidimensional Scaling*. Sage Publications.

[58] Lawrence, J. (1991). "Polytope Volume Computation", *Mathematics of Computation*, Vol.57, No.195, 259-271.

[59] Pickover, C. A. (1988). "Pattern Formation and Chaos in Networks", *Communications of the ACM*, Vol.31, No.2, 136-151.

[60] Marcotorchino, F. (1991). "Seriation Problems: An Overview", *Applied Stochastic Models and Data Analysis*, Vol.7, 139-151.

[61] Martin, A. R. and M. O. Ward (1995). "High Dimensional Brushing for Interactive Exploration of Multivariate Data", *Proceedings of the 6th conference on Visualization '95*. 271.

[62] McCormick, B. H., T. A. Defanti, and M. D. Brown (1987). "Visualization in Scientific Computing", *Computer Graphics*, Vol.21, No.6.

[63] Musser, B. J. (1999). "Extensions to Recursive Partitioning", *Doctoral Dissertation*, University of Minnesota, School of Statistics.

[64] Richardson, M. W. (1938). "Multidimensional Psychophysics", *Psychological Bulletin* 35: 659-660.

[65] Robertson, G. G., J. D. Mackinlay, and S. K. Card (1991). "Cone trees: Animated 3D Visualizations of Hierachical Information", *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems, Information Visualization.* 189-194.

[66] Sagan, H. (1994). *Space-Filling Curves*, Springer-Verlag, New York.

[67] Shneiderman, B. (1992). "Tree Visualization with Tree-maps; A 2D Space-Filling Approach", *ACM Transactions on Graphics*. Vol.11, No.1, 92-99.

[68] Shneiderman, B. (1996). "The eyes have it: A task by data type taxonomy for information visualizations", *Proceedings of IEEE Workshop on Visual Language*, 336-343.

[69] Shneiderman, B. and M. Wattenberg (2001). "Ordered Treemap Layouts", *Proceedings of the IEEE Symposium on Information Visualization 2001*.

[70] Streng, R. (1991). "Classification and Seriation by Iterative Reordering of a Data Matrix", *In Classification, Data Analysis, and Knowledge Organization: Models and Methods with Applications*, 121-130, edited by Bock, H. H., and Ihm, P. Springer-Verlag, New York.

[71] Sun, L., S. Smith, and T. P. Caudell (2003). "A Low Complexity Recursive Force-Directed Tree Layout Algorithm Based on the Lennard-Jones Potential", *University of New Mexico Technical Report*: EECE-TR-03-001.

[72] Sun, R. and D. B. Carr (2005). "Hexagonal Layouts for Hierarchical Structures", *Proceedings of Joint Statistical Meetings, August, 2005.*

[73] Tamassia R., G. D. Battista, and C. Batini (1988). “Automatic Graph Drawing and Readability of Diagrams”, IEEE Transactions on Systems, Man and Cybernetics, Vol.18, No.1, 61-79.

[74] Tavanti, M. and M. Lind (2001). “2D vs 3D, Implications on Spatial Memory”, *Proceedings of the IEEE Symposium on Information Visualization 2001*.

[75] Torgerson, W. S. (1958). *Theory and Methods of Scaling*. John Wiley, New York.

[76] Toronen, P., M. Kolehmainen, G. Wong, and E. Castren (1999). “Analysis of Gene Expression Data Using Self-Organizing Maps”, *Federation of European Biochemical Societies Letters* 451, 142-146.

[77] Trosset, M. (2005). “Visualizing Correlation”, *Journal of Computational and Graphical Statistics*, Vol.14, No.1, 1-19.

[78] Trosset, M. (2005). “Representing Clusters: K-Means Clustering, Self-Organizing Maps, and Multidimensional Scaling”, *Technical Report*. Department of Mathematics, College of William & Mary, P.O. Box 8795, Williamsburg, VA 23187.

[79] Tufte, E. (1983). *The Visual Display of Quantitative Information*. Cheshire Press.

[80] Tufte, E. (1990). *Envisioning Information*. Cheshire Press.

[81] Tufte, E. (1997). *Visual Explanations*. Cheshire Press.

[82] Van Wijk, J. J. and H. van de Wetering (1999). “Cushion treemaps: Visualization of hierarchical information”, *Proceedings of the IEEE Symposium on Information Visualization.* 73-78. IEEE Computer Society.

[83] Venables, W. N. and B. D. Ripley (1999). *Modern Applied Statistics with S-Plus, 3rd Edition.* Springer, New York.

[84] Vernier, F. and L. Nigay (2000). “Modifiable treemaps containing variable-shaped units”, *Extended Abstracts of the IEEE Symposium on Information Visualization.* IEEE Computer Society.

[85] Vesanto, J. (2002). “Data Exploration Process Based on the Self-Organizing Maps”, Doctoral Dissertation, Helsinki University of Technology.

[86] Ware, C. (2004). *Information Visualization, Second Edition: Perception for Design, 2nd Edition.* Morgan Kaufmann.

[87] Ware, C. and G. Franck (1996). "Evaluating Stereo and Motion Cues for Visualizing Information Nets in Three Dimensions", *ACM Transactions on Graphics*, Vol.15, No.2, 121-140.

[88] Ware, C. and P. Mitchell (2008). "Visualizing Graphs in Three Dimensions", *ACM Transactions on Applied Perception*, Vol.5, No.1, 2:1-15.

[89] Wattenberg, M. (1999). "Visualizing the stock market", *Extended Abstracts on Human Factors in Computing Systems.* 188-189. ACM Press.

[90] Wattenberg, M. (2005b). "A Note on Space-Filling Visualizations and Space-Filling Curves", *IEEE Symposium on Information Visualization.* IEEE.

[91] Wilkinson, L. (1999). *The Grammar of Graphics (Statistics and Computing)* Springer Verlag.

[92] Wills, G. J. (1998). "An Interactive View for Hierarchical Clustering", *IEEE*, 26-31.

[93] Wills, G. J. (1999). "Niche Works – Interactive Visualization of Very Large Graphs", *Journal of Computational and Graphical Statistics*, Vol.8, No.2, 190-212.

[94] Young, F. (1985). "Multidimensional Scaling", Kotz-Johnson (Ed.) *Encyclopedia of Statistical Sciences*, Vol.5, John Wiley & Sons, Inc.

CURRICULUM VITAE

Ru Sun received her Bachelor of Science from Jilin University in 1993 and Master of Science in 1996 from the same university in P. R. China, major in Computational Mathematics & Applied Software. She came to the United States in 1997 and received a Master of Science from George Mason University in 2000. She was employed as a business consultant at the Quantitative Economics and Statistics group at Ernst & Young in 2002 and has been working there since. The projects she has been involved in are in various statistical areas: data analysis and visualization, sampling, forecasting, modeling, survey, statistical testing, etc.