The novel application of dynamic graphics to unsupervised learning, graphs, and cycle clustering

by

Zachary Thomas Cox

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Electrical Engineering

Program of Study Committee:
Julie Dickerson, Major Professor
Daniel Ashlock
Daniel Berleant

Iowa State University

Ames, Iowa

2002

Graduate College
Iowa State University

This is to certify that the master's thesis of

Zachary Thomas Cox

has met the thesis requirements of Iowa State University

____
Major Professor

____
For the Major Program

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Dynamic graphics are a collection of data visualization techniques that employ interaction and real-time updating to enhance graphical displays. Traditionally, dynamic graphics have been used to augment classic statistical plots, such as scatter plots. This thesis describes the novel application of dynamic graphics to explore several different areas: traditional unsupervised learning algorithms, graphs, and new applications of clustering algorithms to graphs.

Unsupervised learning algorithms are automated methods for exploring and finding the interesting aspects of a high-dimensional data set. The grand tour is a multivariate data visualization technique used to observe the unsupervised learning algorithms. Making the plots interactive and utilizing linked brushing presents more information to the user.

A graph consists of nodes and edges. Graphically, nodes are small shapes like circles and edges are shown as lines connecting the nodes. Interaction with the visualization allows for structure modification such as node addition/deletion, edge addition/modification/deletion, and node and edge figure placement. The nodes and edges of a graph can have multiple properties, or variables, associated with them, and these values can be shown in the visualization by mapping them to visual attributes of the nodes and edges, like node shape or edge color.

A path is a sequence of nodes and edges of a graph and a cycle is a path that leads back to the starting node. Many of these cycles in a graph will contain some of the same nodes and edges. Groups of similar cycles can be found by using a clustering algorithm on the cycles of the graph. A suitable distance metric for comparing cycles and a definition of the median of a set of cycles are required by the clustering algorithm. Once the clusters of similar cycles are found, dynamic graphics methods can be used to visualize them effectively.

GENERAL INTRODUCTION

Introduction

Dynamic graphics are a collection of data visualization techniques that employ interaction and real-time updating to enhance graphical displays. Allowing the user to interact with the display and updating that display using animation can show much more information than static plots. Traditionally, dynamic graphics have been used to augment classic statistical plots, such as scatter plots. These techniques can be applied to other domains as well, offering new visualization and analysis methods. This thesis describes the novel application of dynamic graphics to explore several different areas: traditional unsupervised learning algorithms, graphs, and new applications of clustering algorithms to graphs.

Unsupervised learning algorithms are automated methods for exploring and finding the interesting aspects of a data set. These methods are typically used on high dimensional data sets to find cluster structure or reduce dimensionality. Visualization, if used in an analysis of the results at all, has traditionally been limited to static two-dimensional scatter plots. However, dynamic graphics offer several new techniques for more effectively visualizing and understanding unsupervised learning algorithms. As will be shown, these techniques create much more powerful and expressive visualizations than traditional methods used in the pattern recognition and machine learning communities.

Unsupervised learning occurs in the high dimensional space of the data. Visualizing the progress and results of the learning algorithm in the high dimensional space with the data points is important, both to understand how the algorithm works and to validate its results. The grand tour is a multivariate data visualization technique used to observe the unsupervised learning algorithms. Using the grand tour, one can see how the algorithm operates in the high dimensional data space. Static low-dimensional projections do not always provide this level of model validation.

The unsupervised learning algorithms can produce more information than can be effectively conveyed in a single static plot of the data. By making the plots interactive, the user can choose and customize which information is shown. Also, linking the display of several different plots can present more information to the user by showing multiple interactive views of the data.

A graph consists of two sets: a set of objects called nodes, and a set of either ordered or unordered pair wise relations between nodes called edges. Graphically, nodes are shown as small shapes like circles or squares and edges are shown as lines connecting the nodes. If the edges are ordered (or directed), there will be an arrowhead on one end of the line. A graph layout algorithm is used to compute the positions of the node and edge figures based on the structure of the nodes and edges in the graph. This is all that is needed to draw a static picture of a graph.

Basic interaction with the graph visualization allows for graph structure modification such as node addition/deletion and edge addition/modification/deletion. In addition, the node and edge figures themselves can be repositioned without affecting the graph structure. Panning and zooming allow a large graph to be viewed in a small window with different levels of detail. Many graph visualization packages only produce static pictures of nodes and edges. FCModeler, the graph visualization package described in this thesis, allows full interaction with the graph.

Graphs are used to model real-world data, like telephone calls, social interactions, or metabolic networks. Thus, the nodes and edges of a graph can have multiple properties, or variables, associated with them. Each node and edge then has a specific value for each property. These values can be shown in the graph visualization by mapping them to specific visual attributes of the nodes and edges, like node shape or edge color. This new mapping process, also implemented in FCModeler, can be done at run-time by a user and adds a greater amount of information about the graph to the visualization than other available packages.

A path is a sequence of nodes and edges of a graph, built by starting at a node and following an edge to another node, then following a different edge to another node, and so on. A cycle is a path that leads back to the starting node. Depending on the problem domain, cycles can be interesting to analyze and efficient algorithms exist to find all of the cycles in a graph.

Large graphs can produce a very large number of cycles. Many of these cycles will come from similar regions of the graph and will thus contain some of the same nodes and edges. Groups of similar cycles can be found by using a clustering algorithm on the cycles of the graph. A suitable distance metric for comparing cycles and a definition of the median of a set of cycles are required by the clustering algorithm. Once the clusters of similar cycles are found, dynamic graphics methods can be used to visualize them effectively.

Overview of Results

This thesis describes several new applications of dynamic graphics. The first is the visualization of three unsupervised learning algorithms: k-means clustering, fuzzy c-means clustering, and self-organizing maps. The first two algorithms find the cluster structure in a data set while the third algorithm can be used both to find the cluster structure and reduce the dimensionality of the data set. The visualization methods are based on using the grand tour to visualize the algorithm and data in the high dimensional data space and interactive plots to explore the results of the learning algorithms. These visualization methods can also be applied to various other unsupervised learning algorithms.

Second is the use of dynamic graphics with multivariate graph visualization. In particular, the nodes and edges of the graph have multiple categorical variables associated with them. Dynamic graphics methods are used to allow graph structure and graph view modification by making the graph visualization interactive. In addition, mappings are made from values of the node and edge variables to visual attributes of the node and edge figures. These mappings are done in real-time in the visualization, letting the user see the values of the node and edge variables.

Finally, dynamic graphics are used to visualize the results of clustering the cycles of a graph. After the cycles of a graph have been found, self-organizing maps are used to find groups of similar cycles based on their node and edge content. This involves creating a distance metric for cycles and defining the generalized median for a set of cycles. The results are shown in an interactive display, where the user can effectively visualize the groups of similar cycles.

k-Means Clustering

The k-means clustering algorithm iteratively partitions the data set into clusters of data points and calculates a prototype, or mean, for each cluster. The means are shown in the grand tour view as large data points and move through the data space as the algorithm progresses. Since each data point is assigned to one of the clusters, color is used to show the cluster membership of the data points. Figure 1 below shows an example of the k-means visualization method.

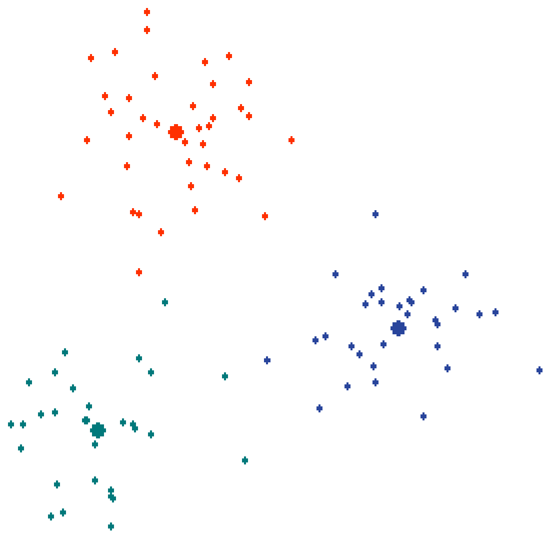


Figure 1. A snapshot of the grand tour view of the k-means clustering algorithm. The cluster means are shown as large data points and the data points are colored according to their cluster membership.

Fuzzy c-Means Clustering

The fuzzy c-means algorithm is similar to the k-means algorithm except that it assigns a membership value in each cluster to each data point, instead of membership in only a single cluster. Thus, each data point belongs to all clusters to different degrees. Again the grand tour is used to visualize the data and the cluster means in the high dimensional data space.

Cluster membership visualization is much more difficult for fuzzy c-means than for k-means, since each data point has a membership value in each cluster. For more than two clusters, showing all of the membership values on each data point would produce a very messy plot. To reduce the amount of information shown to a comprehendible amount, two classes of membership visualization methods were developed: showing membership in a single cluster using some other variable (like color or height) and showing membership in all clusters above a certain threshold value (using convex hulls or translucent glyphs). Figure 2 below shows an example of the first membership visualization method, where each data point is colored based on its membership in the blue (lower left) cluster. Red represents a high membership value in this cluster while light blue represents low membership.

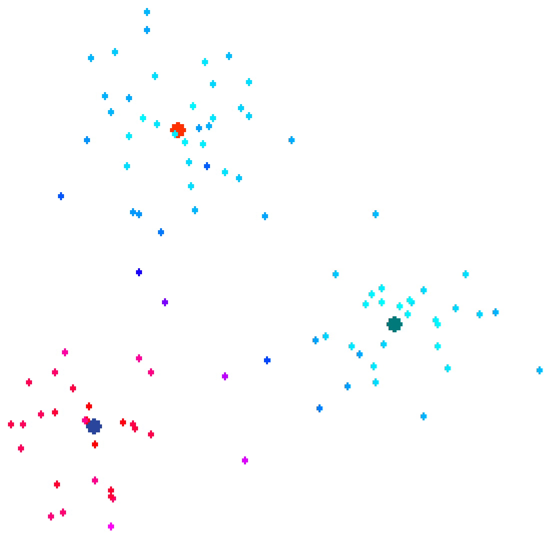


Figure 2. A snapshot of the grand tour view of the fuzzy c-means clustering algorithm. The data points are colored according to their membership in the blue (lower left) cluster, with red representing high membership and light blue representing low membership.

Self-Organizing Maps

The self-organizing map (SOM) is an unsupervised learning algorithm that can be used for clustering and dimensionality reduction. It takes a low dimensional (usually 1-D or 2-D) grid of points and spreads it throughout the high dimensional data space, approximating the data distribution. The points on the map will be pulled to the natural clusters of data points, and the data points can then be projected from the high dimensional data space to the low dimensional space of the map.

The grand tour is used again to visualize the data and the map in the high dimensional data space. This view provides trust to the user that the map does indeed approximate the distribution. A view of the low dimensional space is also used to visualize the map. The two views are linked, so that interacting with one view causes changes to occur in the other view. For example, the data points can be interactively projected onto the map by brushing them in the tour view with different colors. The corresponding map units are then automatically brushed with the same color. Figure 3 shows an example of the SOM visualization methods.

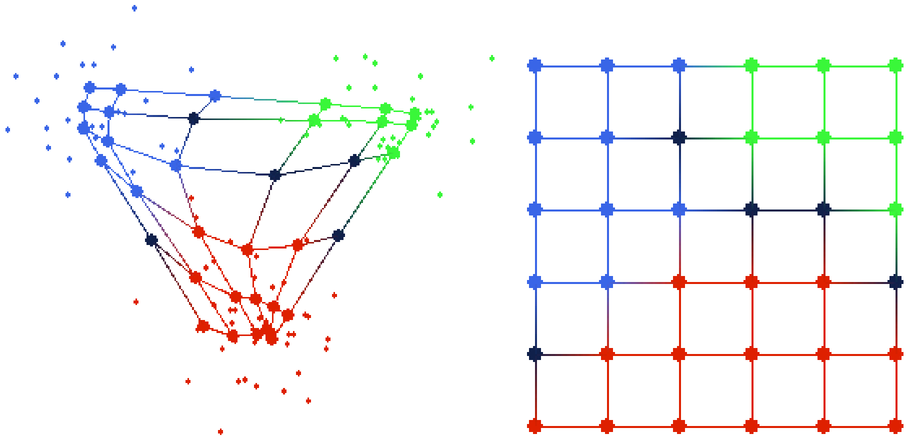


Figure 3. Example of the SOM visualization methods. The grand tour view (left) shows the data and the map in the high dimensional space, while the map view (right) shows the data projected into the low dimensional space. The data points and map were colored interactively based on the clusters in the data.

Graph Visualization

Graph visualization is a well-studied topic, and many commercial software packages exist that are used for visualizing graphs. The main contribution of the author to this area is the creation of the FCModeler software. FCModeler is a Java-based software package used to visualize, analyze, and simulate metabolic networks. It can be used as a more general tool however, for interactively visualizing multivariate graphs in general. The visualization supports full interaction, including graph structure modification, graph view modification, panning, and zooming.

FCModeler also provides the capability of mapping values of specific node and edge variables to visual attributes of the node and edge figures, respectively. These mapping rules allow such complex sentences such as “If the node is a molecule and its location is the plastid, then its shape is an ellipse”. The system dynamically determines which values of the various node and edge variables exist, and lets the user map them to any specific visual attribute, such as node shape, node fill color, edge color, edge dash pattern, etc. Figure 4 below shows an example of a graph in FCModeler.

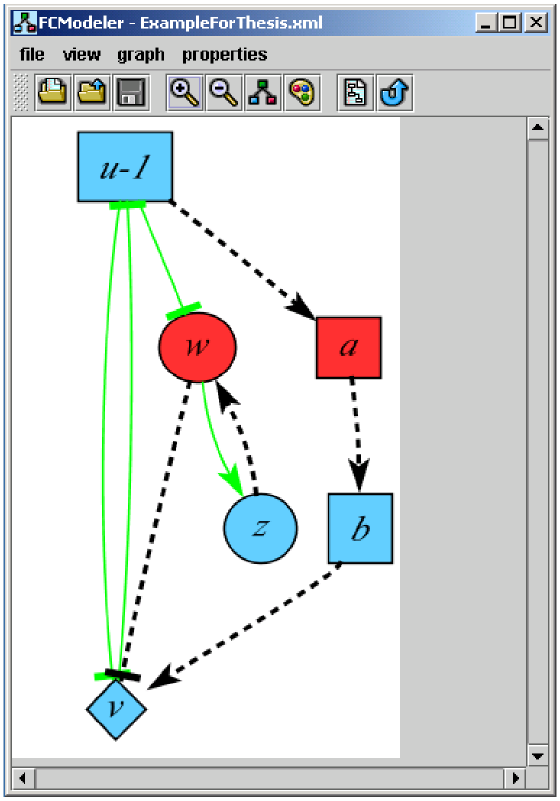


Figure 4. Example of a directed graph shown in FCModeler. The different visual attributes convey different values of the node and edge variable associated with the graph.

Cycle Clustering and Visualization

Cycles obtained from directed graphs may be similar to each other based on their node and edge content. Thus, clustering algorithms may be used to find natural groups of similar cycles. Prior to clustering, a distance metric and generalized set median must be defined for the objects to be clustered. Several possible representations for the cycles are discussed, including strings, graphs, and weighted sets. Weighted sets are chosen because of the simplicity of the associated metric and median. Self-organizing maps (SOM) are used to find the clusters of cycles obtained from a directed graph. This innovative approach is implemented in FCModeler and uses the JSOMap package for self-organizing maps.

The results of the SOM clustering are shown in a grid of graph views, each representing a cluster of cycles discovered by the SOM algorithm. Clicking on one of these small graph views shows the cluster of cycles in a larger, more easily understandable window. These interactive visualizations effectively show the results of the SOM clustering so they can be analyzed by the user.

Thesis Organization

This thesis consists of a general introduction chapter, four research papers, and a general conclusions chapter. References for all chapters are included in a comprehensive reference listing immediately following the conclusions chapter. The research papers are as follows:

Publication 1. Cox, Z., J. A. Dickerson, D. Cook. (2001). Visualizing Membership in Multiple Clusters after Fuzzy c-Means Clustering. Proceedings of Visual Data Exploration and Analysis VIII, San Jose, CA.

This paper describes the use of dynamic graphics techniques to visualize the k-means and fuzzy c-means clustering algorithms. The grand tour was used to visualize the positioning of the means by both algorithms in the high dimensional data space. Color of the data points was used to visually distinguish the clusters calculated by the k-means algorithm. Four new methods of visualizing the memberships of the data points calculated by the fuzzy c-means algorithm were also presented. The author conceptualized the fuzzy c-means visualization methods, implemented the methods in software, and was principal author of the paper.

Publication 2. Cox, Z. (to be submitted). "Visual Exploration of Self-Organizing Maps Using the Graph Tour and Linked Brushing." Journal of Statistical Software.

This paper describes the use of dynamic graphics techniques to visualize self-organizing maps. The grand tour was used to visualize how the algorithm progressively positioned the map in the high dimensional space of the data. The grand tour view was linked to a view of the map in feature space to allow the user to interactively explore the mapping between input space and feature space. The author designed the visualization methods, implemented them in software, created an object-oriented self-organizing map package, and was principal author of the paper.

Publication 3. Dickerson, J. A., D. Berleant, Z. Cox, W. Qi, D. Ashlock, E. Wurtele, A. W. Fulmer. (to be published in 2002). Creating and Modeling Metabolic and Regulatory

Networks Using Text Mining and Fuzzy Expert Systems. Computational Biology and Genome Informatics. C. H. Wu, P. Wang and J. T. L. Wang, World Scientific.

This book chapter describes the Gene Expression Toolkit. This toolkit will aid in the analysis and comparison of large microarray, proteomics and metabolomics data sets. It also aids in the synthesis of the new test results into the existing body of knowledge on metabolism. The user can select parameters for comparison such as species, experimental conditions, and developmental stage. The author designed and implemented FCModeler, a software tool used to visualize, analyze, and simulate metabolic networks, which is a significant part of the Gene Expression Toolkit. The author also contributed significantly to the sections of the book chapter describing FCModeler.

Publication 4. Cox, Z. (to be submitted). "Clustering Cycles Using Self-Organizing Maps." Journal of Statistical Software.

This paper describes the use of self-organizing maps to cluster the cycles obtained from a directed graph. It presents an overview of the different cycle representations, including strings, graphs, and weighted sets. Various distance metrics and generalized set medians for the different representations are also reviewed. Once the clustering is complete, the results are visualized in informative, interactive graph visualizations. The author researched the different cycle representations, implemented the clustering using weighted sets, created the visualization methods, and was principal author of the paper.

Literature Review

Dynamic Graphics

The use of interactive computer graphics in statistical analysis dates back to the 1970's. The first major interactive statistical graphics system was PRIM-9 (Fisherkeller, Friedman et al. 1974) which featured dynamic tools for projecting, rotating, isolating, and masking multidimensional data in up to nine dimensions. PRIM-9 and other related systems are described in (Cleveland and McGill 1988). (Becker, Cleveland et al. 1988; Buja, Cook et al. 1996) present good general introductions to dynamic graphics techniques.

The grand tour is a dynamic graphics method for visualizing high dimensional data, and is first described as a data analysis tool in (Asimov 1985). (Cook, Buja et al. 1995) guide the projections of the grand tour using projection pursuit, and (Cook and Buja 1997) present methods for manually controlling the variable contributions to the projection.

Linked brushing is the general method of connecting multiple views of a data set, so that the appearance of data points in all of the plots is the same. (McDonald 1982; Becker and Cleveland 1987) provide descriptions of interacting with one of a collection of plots and having the other plots updated in real-time to show the changes.

Unsupervised Learning Algorithms

Unsupervised learning is a well-studied field, with roots in pattern recognition, machine learning, artificial intelligence, and density approximation. According to (Cherkassky and Mulier 1998), unsupervised learning methods can be used in four different ways: data/dimensionality reduction, interpretation of the data set, predictive modeling for future samples, and preprocessing for supervised learning.

There are many good references for the k-means clustering algorithm; any text on pattern recognition or machine learning will most likely contain a description. (Nadler and Smith 1993; Ripley 1996) provide good introductions to unsupervised learning algorithms. The fuzzy c-means clustering algorithm is also widely studied, with the classic reference (Bezdek 1981). Self-organizing maps (also called Kohonen maps) were created by Kohonen and he describes them in (Kohonen 1984; Kohonen 1995; Kohonen 1997; Kohonen 2001).

Analysis of Unsupervised Learning Algorithms

In general, there are two traditional ways to analyze the results of clustering algorithms such as k-means and fuzzy c-means. The first involves calculating some validity index that shows how well the clustering algorithm partitioned the data (Jain and Dubes 1988). Usually the clustering algorithm is run on the data several times with different parameters, and then the validity index for each run shows which partition is “the best”. In the second method, the traditional pair wise scatterplot shows the cluster means and different glyphs are drawn for the data points to represent the different clusters. Visualization can provide confidence in the clustering algorithm and may suggest different parameters to try for more effective results.

(Fayyad, Grinstein et al. 2002) is the only real text covering more in-depth visualization of data mining algorithms, but it doesn’t really give clear examples of how to visualize the many different algorithms. They mainly describe the various multivariate visualization methods currently available and present “requirement specification” papers on what successful data mining visualizations should be like. The paper by (Thearling, Becker et al. 2002) in the previous text discusses model visualization, but only for supervised learning methods such as decision trees and tables. The visualizations they describe also have nothing to do with viewing the models in the high dimensional data space.

Little work has been done on visualizing the fuzzy c-means clustering algorithm. The Fuzzy Logic Toolbox (MathWorks 2002), an add-on to the popular technical computing environment MATLAB, uses 2-D scatterplots to show the results of the fuzzy c-means clustering algorithm. These scatterplots show the cluster means and use color to show the cluster each data point belongs to the most.

FCLUST (Egan, Krishnamoorthy et al. 1998), a visualization tool for fuzzy clustering, shows 2-D and 3-D scatterplots of the data set and uses color based on the highest membership of the data points. It also highlights data points with membership values above a certain threshold, which is similar to the threshold method described in this thesis. However, none of the inherent problems of their method are discussed, nor are any potential solutions suggested.

More research has been done related to SOM visualization than that of the k-means or fuzzy c-means clustering algorithms. This is probably due to the fact that the SOM itself can be used as a data mining visualization tool, projecting high dimensional data to a 2-D display. Much of the work has focused on the 2-D map display, with little attention paid to the map in the high dimensional data space.

(Vesanto 1999) provides an overview of several state-of-the-art SOM visualization methods, which are all based on the 2-D map view. In one method, connections between the nodes of the map are colored based on the distance between the nodes’ models in input space. The larger the distance between the models, the darker the connections are colored on the map. Thus, clusters are dense groups of white connections on the map, with black connections separating them. Also, the size of a node in the map view can be made proportional to the average distance from its model to all of its neighboring nodes’ models. This plot shows basically the same information as the first, where clusters are now shown as groups of small nodes with large nodes separating them. A third method they describe uses component planes, where each node of the map is colored based on its model's value of a single variable. This visualization can show where high or low values of certain variables are located on the map. Data histograms are another method, where the nodes are colored based on how many data points are projected to them.

(Kaski, Venna et al. 1999) describe a method for assigning colors to the nodes of the map in order to accentuate the cluster structure of a data set. This is more or less an extension of several methods described in (Vesanto 1999), where an entire color space is used instead of just grayscale or glyph size. Nodes of the map with similar models in the input space are colored similarly, while nodes with dissimilar models are colored differently. In the 2-D map view, clusters can then be seen as groups of nodes with similar color.

(Himberg 1998) produces two different views of the SOM and links the coloring of the nodes of the map between the two views. The two views used are Sammon's projection (Sammon 1969) and the traditional 2-D map view. The node colors are calculated from the map unit's 2-D positions in either plot and are static. This coloring is meant to show the user how the map was positioned in the high dimensional data space. The Sammon's projection view shows only the projected map, with no data points.

Note that except for (Himberg 1998), all of these visualizations show a single static view of the map in feature space. There is absolutely no analysis of the map's position in the data space, aside from the Sammon's projection view. And in that view, the data points are not even shown!

(Thearling, Becker et al. 2002) discuss the importance of using visualization to promote understanding and trusting of the model. Clearly, the aforementioned SOM visualization methods provide neither, since there is no way to determine if the map actually matches the distribution of the high dimensional data. Using the grand tour to visualize the map with the data and interactively linking that view with the traditional map view provides the necessary trust (or, for that matter, distrust) in the model.

VISUALIZING MEMBERSHIP IN MULTIPLE CLUSTERS AFTER FUZZY C-MEANS CLUSTERING

A paper published in the Proceedings of Visual Data Exploration and Analysis VIII

Zach Cox[1], Julie A. Dickerson[1], and Dianne Cook[2]

Abstract

Cluster analysis is an exploratory data mining technique that involves grouping data points together based on their similarity. Objects or data points are often similar to points in more than one cluster; this is typically quantified by a measure of membership in a cluster, called fuzziness. Visualizing membership degrees in multiple clusters is the main topic of this paper. We use Orca, a Java-based high-dimensional visualization environment, as the implementation platform to test several approaches, including convex hulls, glyphs, coloring schemes, and 3-dimensional plots.

Keywords: Clustering, fuzzy c-means, visualizing uncertainty, Orca, Java

Introduction

Clustering algorithms attempt to divide a data set into meaningful groups or clusters. These algorithms start with a set of data points and assign them to groups such that the similarity between data points within a group is much higher than the similarity between data points in different groups. These clusters represent some kind of grouping within the data set. For example, clustering has been used to find patterns in RNA expression (Eisen, Spellman, et al 1998), image segmentation (Krishnapuram and Keller 1994), and causes of ischemic stroke (Dickerson, Matalon et al. 1998).

1. Electrical and Computer Engineering Department, 2215 Coover Hall, Iowa State University, Ames, IA, 50011.
2. Statistics Department, 325 Snedecor Hall, Iowa State University, Ames, IA, 50011.

Many data sets do not have separate, distinct clusters. These sets contain data points that don't necessarily fit well into only a single cluster. Instead, these intermediate cases are described to a certain degree by several different clusters. As a simple example, consider a set of apples. Suppose that a clustering algorithm divided this set into two clusters: red apples and green apples. There are probably a number of apples in the red group that have some amount of green on their skins, likewise there are probably some green apples with red on their skins. Depending on the actual data set, these two clusters could blend together smoothly instead of being split into two independent groups. Taking this indistinct nature of the data set into consideration may be a desirable quality of the clustering algorithm used to analyze it.

One such algorithm is the fuzzy *c*-means (FCM) clustering algorithm (Bezdek 1981). The FCM algorithm allows each data point to belong to all clusters simultaneously instead of restricting each data point to belonging to only one cluster. The degree to which a data point belongs to a cluster is shown in its membership value for that cluster. In other words, the membership value expresses how well the data point is described by the cluster. In this way, an apple can belong to both the red group and the green group, and its membership in each group depends on the red-ness and green-ness of the apple.

The membership values obtained from the FCM algorithm are important to analyze. They provide a quantitative description of how well each data point fits in with each cluster. Using data visualization techniques to perform this analysis is the topic of this paper, and we describe several useful methods for doing this. The data visualization toolkit Orca[5] is used to test the different membership visualization methods.

The FCM algorithm will be presented in more detail in Section 2. The visualization environment used to test the different methods of membership visualization, Orca, is described in Section 3. The membership visualization methods themselves are illustrated in Section 4, and Section 5 discusses their various advantages and disadvantages.

Fuzzy c-Means Clustering

The FCM algorithm minimizes the objective function (Bezdek 1981)

$$J(\mathbf{U},\mathbf{V})=\sum_{k=1}^{n}\sum_{i=1}^{c}(u_{ik})^{m}\|x_k-v_i\|^2 \qquad \text{subject to } u_{ij}\in[0,1] \text{ and } \sum_{i=1}^{c}u_{ik}=1\ \forall\ k \qquad (1)$$

U is the partition matrix that shows to what degree the *k*-th data point x_k belongs to each cluster as measured by its distance from the centroid of the *i*-th cluster, v_i. *m* is a weighting exponent, *c* is the number of clusters and *n* is the number of data points. This well-known algorithm has the steps listed below (Bezdek 1981):

1) Fix the number of clusters *c*, $2 \leq c < n$; choose an inner product metric, such as the Euclidean norm and the weighting exponent *m*, $1 \leq m < \infty$. Initialize the partition *U(0)*.

2) Calculate the *c* fuzzy cluster centers $\{\mathbf{v}_i^{(l)}\}$using

$$\mathbf{v}_i = \frac{\sum_{k=1}^{n} (u_{ik})^m \mathbf{x}_k}{\sum_{k=1}^{n} (u_{ik})^m} \tag{2}$$

3) Update the partition matrix *U(l)* and $\{\mathbf{v}_i(l)\}$.

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{\left\| \mathbf{x}_k - \mathbf{v}_i \right\|}{\left\| \mathbf{x}_k - \mathbf{v}_j \right\|} \right)^{2/(m-1)}} \tag{3}$$

4) If $\left\| \mathbf{U}^{(l+1)} - \mathbf{U}^{(l)} \right\| \leq \varepsilon$ then stop, otherwise repeat steps 2 through 4.

Using the above steps, the FCM algorithm assigns each data point x_i a membership value $u_{i,k}$ in each cluster *k*. The membership values are in the range [0.0, 1.0], and all of the membership values for a data point must sum to 1.0. In the limiting case where the membership values are binary (0 or 1), the FCM algorithm reduces to the K-Means clustering algorithm (Duda and Hart 1973). The FCM algorithm also calculates the centroids, or means, of each cluster. It is these two pieces of information, the membership values $u_{i,k}$ and cluster means v_k, that are used to create the membership visualization.

Orca

Orca is a toolkit for producing useful views of data (Sutherland, Rossini et al. 2000). It is written entirely in Java and provides basic functionality for data visualization such as data file parsing, data processing, familiar plots, and user interface management. The software is structured by a pipeline architecture, in which the different pipe sections perform different aspects of data processing and rendering. For example, a typical pipeline starts with a data source pipe, which is responsible for reading the data from a file into an internal array. This raw data (which is assumed in this project to be in arbitrary p-dimensions) is then passed to a standardization pipe where each variable is standardized, for instance to the range [-1, 1] or [0, 1]. The p-dimensional data may now be projected to a lower d-dimension for some type of display (typically $d = 2$ for a computer screen, or $d = 3$ for a virtual reality display). The d-dimensional data is then converted to display coordinates and rendered for the user in a specific plot. The aforementioned pipeline is a basic, typical setup completely provided by core Orca classes.

This core package is also built heavily on Java Interfaces, allowing Orca to be extended easily by end-users to create their own data processing and custom renderings. By providing a few specified methods these custom classes plug-in to the basic API, and gain access to the internal data structures of Orca. This saves an immense amount of development time, as the researcher can focus on specific experimental ideas instead of the rudimentary tasks done by the existing Orca pipe sections. Orca has been used to analyze multivariate time and space measurements in relation to studying the El Nino effect from buoys in the Pacific Ocean (custom time and spatial views of the data were created). It has also been used for research on interaction with graph data, where the nodes and edges have multiple variables.

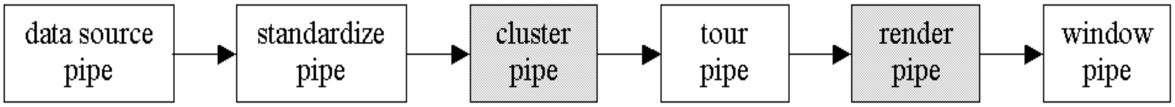


Figure 5. Orca pipeline for the membership visualization project. Custom pipe sections are shaded gray.

We created two custom pipe sections to experiment with membership visualization methods: a clustering pipe and a rendering pipe. The resulting pipeline is shown in Figure 5. The clustering pipe receives the coordinate-free data from the standardization pipe and performs the FCM clustering algorithm on it. This pipe also provides the coordinates of the means and the membership of the data points after each step of the algorithm to pipes farther down the line. The second custom pipe, the rendering pipe, extends the original scatter plot pipe provided by Orca. This new scatter plot uses the information provided by the clustering pipe to implement the various membership visualization schemes. The basic functionality of Orca was used to greatly simplify the design and coding of this project.

Orca also features a Grand Tour engine (Buja, Cook et al. 1996; Swayne, Cook et al. 1998). Since the data to be analyzed is in arbitrary *p*-dimensions, a way of projecting it down to 2-dimensions is needed for rendering in the scatter plot. This projection is done by the grand tour. It creates 2-dimensional projections of the *p*-dimensional data, and rotates smoothly between the projections. The smooth rotation creates a tour, or movie, of the data, which is very useful in determining the structure and features of a multivariate data set.

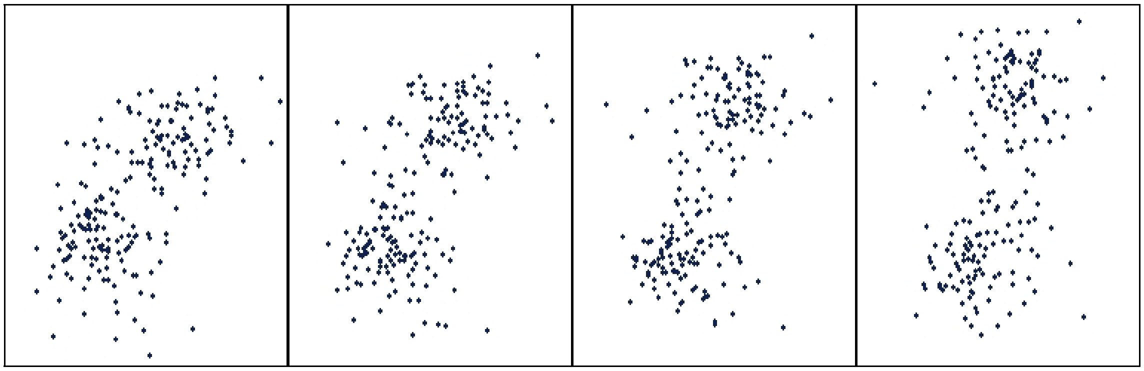


Figure 6. Example of a Grand Tour sequence.

Figure 6 shows four successive snapshots of a Grand Tour running on a data set of 200 data points in 6 variables. Although somewhat difficult to see in static images, the points toward the top of the plot are moving up and left, while the points toward the bottom are moving down and right. We use the grand tour in this project to see not only the data from different angles, but the membership information as well.

Once the clustering algorithm finishes, Orca can use the cluster means to produce a more informative tour over the data set. The basic touring algorithm (described above) works in the *p*-dimensional space of the data, randomly projecting it down to two dimensions. However, the tour can project from the lower dimensional space defined by the cluster means, once they are obtained, instead of from the *p*-dimensional space of the data. This method tends to produce better views of the clusters, because any of the *p* variables that do not contribute to the structure of the clusters are left out. Although this idea of a cluster guided tour is not new, its previous implementations have been less than adequate. The method used in Orca is mathematically correct, and we intend to actively pursue research on this subject in the future.

Membership Visualization Methods

We developed a number of different visualization methods to explore the membership information obtained from FCM clustering. These methods build upon the Grand Tour / scatter plot combination of Orca by adding additional graphics to the scatter plot. The graphics are derived in various ways from the membership values of the data points. Figure 7 shows a hierarchy of the more successful methods, which are described in the following sections.

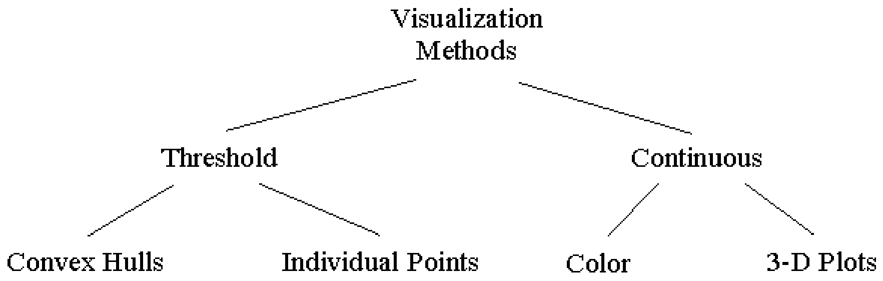


Figure 7. Hierarchy of membership visualization methods.

Threshold

The threshold method begins by finding a subset S_j of the data points x_i that have membership values $u_{i,j}$ above a threshold value t for each cluster j:

$$x_i \in S_j \text{ iff } u_{i,j} \geq t, \forall i,j$$

The subset of data points S_j can then be visually distinguished in some manner from the data points not in S_j. This allows a kind of membership boundary for each cluster to be displayed, in that any data points within the boundary belong to that cluster more than the threshold value. The description thus far of the threshold method is fairly vague, and several more specific implementations can be obtained from it. Two such implementations were carried out in this project: convex hulls and individual points.

Convex Hulls

The convex hull of a set of points in two dimensions is defined as the smallest convex polygon that encloses all of the points. This can be thought of as a rubber band stretched around the set of points, and released to conform to the shape of their outline. Since the 2-D coordinates of the points in S_j are known (from the Grand Tour), the convex hull of this subset can be easily calculated. For each frame of the tour, the Quickhull algorithm (O'Rourke 1998) is used to calculate the convex hull of each S_j. The hull is then rendered on the scatter plot as a translucent polygon.

Figure 9 (which appears on the last page of this paper, along with all other color figures) shows the threshold method using convex hulls for several different threshold values. The large circles represent the cluster means, and are rendered using different colors for clarity. The data points are colored based on the cluster they belong to the most. Similarly, the convex hulls are colored according to the cluster's membership boundary they represent.

An important point to note is that the data sets used in this study typically contain 3 or more variables. Thus, they exist in a higher order space than can be adequately represented on paper (which is of course 2-dimensional). All of the figures used in this paper are simply snap shots of a running tour, and show 2-dimensional projections of a *p*-dimensional data set. These 2-dimensional scatter plots are therefore not fully representative of the overall structure of the data, and this fact should be remembered when viewing the figures.

Individual Points

Instead of creating a filled-in region to differentiate data points with memberships above the threshold from those below it, the individual points method changes the glyph used to represent these data points. Stated more mathematically, it uses a different glyph for the data points in S_j than data points not in S_j. There are many choices for this new glyph, including a different shape (square as opposed to circle), different size, etc., as long as it is visually different from the glyph used for data points not contained in S_j. We chose to implement this method using a slightly larger translucent circle superimposed over the data point, so that it is not too distracting and multiple circles can be combined since they are translucent.

The individual points visualization method is illustrated in Figure 10. The data set and threshold values are the same as those used in Figure 9 for comparison. Again, the data points are colored according to the cluster they belong to the most. A translucent circle is then rendered on top of the data points for each cluster in which they have a membership value above the threshold. Thus, there could be multiple translucent circles on each data point if that point has membership values in several clusters above the threshold. This can be seen in Figure 10 (d), where the data point at the top right of the red cluster has both red and green translucent circles.

Continuous

The continuous membership visualization method is not restricted to either showing membership information or not showing the information, as the threshold method is. Instead, it allows each data point to visually express its membership value in a single cluster. In this way, the variation of membership values across the entire data set can be easily seen. There are obviously many ways to visually express the membership values, and we now present several useful methods for doing this.

Color

The color of the data point is one way to show its membership in a cluster. Two different colors, c_0 and c_1, are assigned to the minimum and maximum values of the membership value (0.0 and 1.0 respectively). Then the color of the data point is found by interpolating between c_0 and c_1 in some manner. The term interpolation is used loosely in this context, and does not necessarily refer to simple linear interpolation between two colors (as in color gradient calculation). We investigated several methods of interpolation, and a simple temperature scheme provided the best results.

This temperature method assigns hot and cold colors to the data points based on their membership values. Since the colors representing hot and cold are not well defined, there is room for interpretation in their selection. However, it is generally understood that red stands for hot, and blue stands for cold. Thus, the red, green, and blue (RGB) values for c_0 and c_1 were assigned as follows:

$$c_0 = [0.0\ 1.0\ 1.0]$$

$$c_1 = [1.0\ 0.0\ 0.0]$$

These RGB values produce cyan (a light blue color) for c_0 and red for c_1. The interpolation to obtain the color of the data point, c, is then implemented as three piecewise linear interpolations, based on the membership value $u_{i,j}$.

$0.0 \leq u_{i,j} \leq 0.33$ interpolate between [0.0 1.0 1.0] and [0.0 0.0 1.0] to get c
$0.33 < u_{i,j} \leq 0.67$ interpolate between [0.0 0.0 1.0] and [1.0 0.0 1.0] to get c
$0.67 < u_{i,j} \leq 1.0$ interpolate between [1.0 0.0 1.0] and [1.0 0.0 0.0] to get c

This amounts to starting with c_0 and decreasing the green value in the first interval, then increasing the red value in the second interval, and finally decreasing the blue value in the third interval to obtain c_1. The temperature coloring method is shown in Figure 11, with the red cluster selected. Note that the data points near the red cluster's mean have hot colors, and data points farther away have cooler colors.

3-D Plots

Extending the 2-D scatter plot to three dimensions is another way to show the membership values of the data points. This can be thought of as turning the points in the scatter plot into lines perpendicular to the 2-D plane of the plot (or coming out of the monitor, for example). The height of the lines is directly proportional to their membership value in the selected cluster.

This method obviously adds an extra dimension to the visualization, which the computer monitor cannot directly support. The Java3D package is used in this case to handle the 3-D data. A flat plane is created, which represents the two dimensions of the scatter plot. The data points are then extended into the third dimension using their membership values. The scene can be rotated in all directions using the mouse, to gain a better perspective of the data. Figure 12 shows the same data set as Figure 11, using the 3-D plot method.

Results / Discussion

Like any data visualization, the methods described in this paper all have different advantages and disadvantages based on the structure of the data set. More effort is required to identify these strengths and weaknesses, as this research in visualizing the membership values provided by the FCM clustering algorithm is still young. Some degree of work has been done in this area however, and the results are presented in this section.

Convex Hulls

The use of convex hulls provides a fast, easy way to show the data points contained in S_j. The translucent polygon rendered on the scatter plot represents the p-dimensional convex hull of S_j projected down to two dimensions, and thus quickly shows all of the data points within the membership boundary defined by the threshold value. The hull is also additive to the scatter plot; that is, it still allows the underlying data points to be seen.

An important part of any visualization algorithm used with the Grand Tour is its computation time. The tour algorithm conceptually creates frames of a movie that need to be rendered fast enough to appear smooth to the user. Thus, any computation on a frame of the tour must finish before the next frame needs to be shown. The Quickhull algorithm has an average computation time of O(n log n) (O'Rourke 1998), where n is the number data points in S_j. This algorithm is fast enough to appear real-time to the user, and does not slow down the tour at all.

The major disadvantage of the convex hull method arises from the projection of high dimensional data down to two dimensions. Occasionally, data points with membership values below the threshold are projected inside the convex hull. These data points are then falsely identified as having membership values above the threshold. Figure 13 illustrates this case, showing both the convex hull and individual points methods with the same threshold value. The data points highlighted by blue squares have memberships in the red cluster below the threshold value, however they are projected into the red convex hull by the tour. The individual points method clearly shows that these data points are not contained in S_j by the glyphs used to represent them.

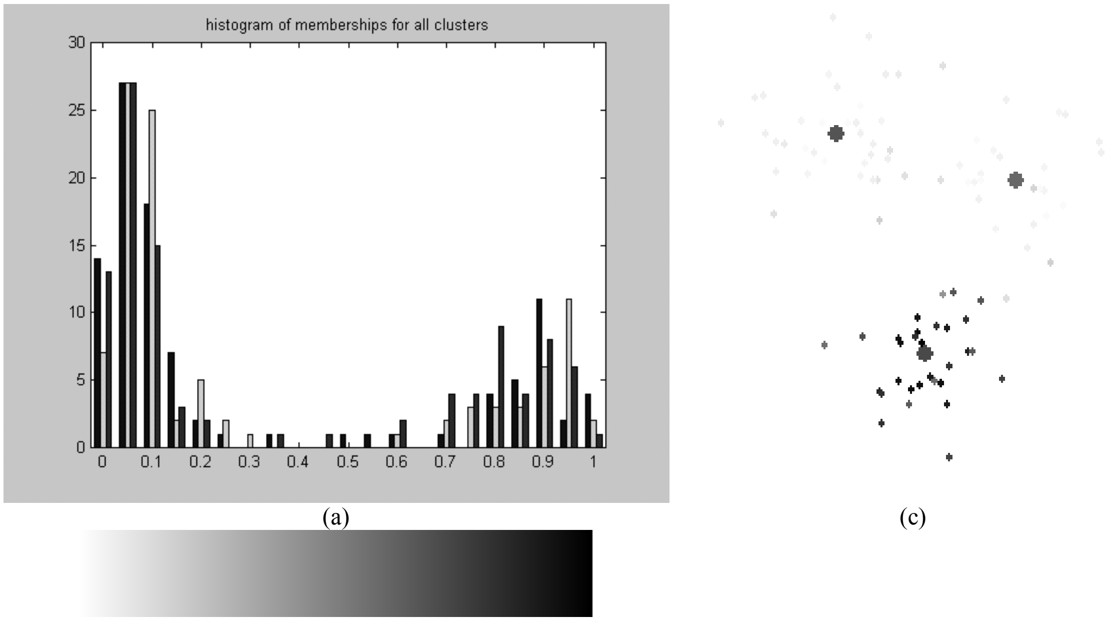
Individual Points

Figure 13, discussed in the convex hull section, demonstrates a strong point of the individual points visualization method. Using this method, there is absolutely no confusion as to which data points are contained by S_j since they are rendered using different glyphs. An implementation of the individual points method could use any glyph to distinguish the data points. This illustrates the flexibility of the method, although only one such glyph was used in this project.

For low threshold values, the individual points method becomes somewhat unusable. Since the different glyphs are rendered for each cluster the data point belongs to more than the threshold value, multiple glyphs are shown at low threshold values. This results in a muddy plot, as shown in Figure 14 (a). This problem is averted by only considering some subset of S_j's when creating the plot. For example, Figure 14 (a) uses all three clusters when creating the membership visualization information, while Figure 14 (b) renders information for only the red cluster.

Color

The color method is useful for visualizing how the membership values in a single cluster vary over the data set, without having to change a threshold value. However, color is a very inaccurate attribute to work with. The exact membership value expressed by a certain color can't easily be identified; only the approximate range of the membership value can be seen. Also, the interpolation used to calculate the color of the data point greatly affects how the membership values appear to change. Figure 8 (a) shows a histogram of the membership values for a data set with three clusters (for each bin there are three bars, one for each cluster: a dark one, a lighter one, and another dark one). This histogram shows that most of the membership values are either below 0.2 or above 0.7. If the simple white to black gradient of Figure 8 (b) is used, there will only appear to be two different colors in the plot: a light gray and black. Figure 8 (c) shows the data set used to generate the histogram of Figure 8 (a) using the interpolation in Figure 8 (b) for the individual points method.



(a)

(c)

(b)

Figure 8. (a) Histogram of data set with three clusters. (b) Simple white to black gradient. (c) Data set used to create histogram in (a), using gradient in (b) for individual points visualization method.

3-D Plots

The 3-D plots provide several good features for visualizing the membership values. The different membership values are easy to see, since the height of the line is directly proportional to the membership value. There is no ambiguity from color as in the previous method. Also, the height of the lines is an exact quantity; the plot could contain axes and the membership value could be readily obtained from the plot.

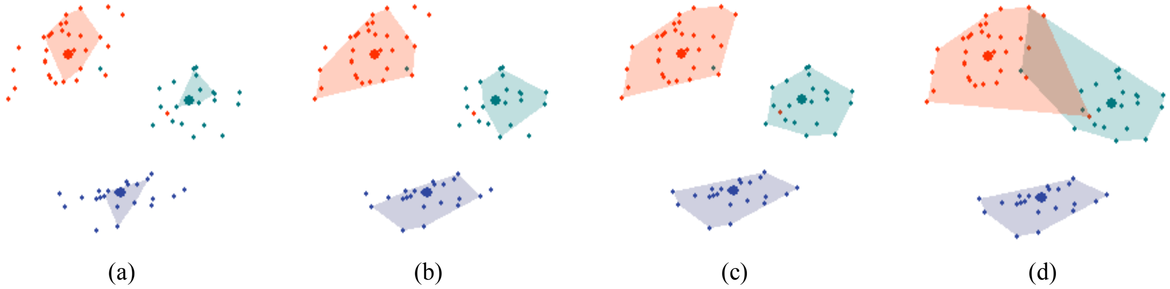
The main drawback of the 3-D plots method stems from the fact that it is 3-dimensional. In order to see the heights of the lines, the scatter plot plane must be rotated. It then becomes non-orthogonal to the viewer. This rotation results in a slightly skewed view of the scatter plot plane. However, this typically does not affect the quality of the visualization, and the general structure of the 2-D scatter plot remains.

Conclusion

Through this research, we introduced several methods for visualizing the membership information provided by the FCM clustering algorithm. These methods can provide great insight into the cluster structure of a data set. They allow the user to actively explore how the data set was clustered, using a mix of well-defined and novel data visualization techniques. Some weaknesses of these methods were also pointed out, as it is important to realize the scope in which they are useful. Investigation on this topic is ongoing, and will most definitely yield better-defined scenarios in which the different methods are most valuable. We hope that the material presented in this paper may provide other researchers with new tools to better analyze and understand the results of fuzzy *c*-means clustering.

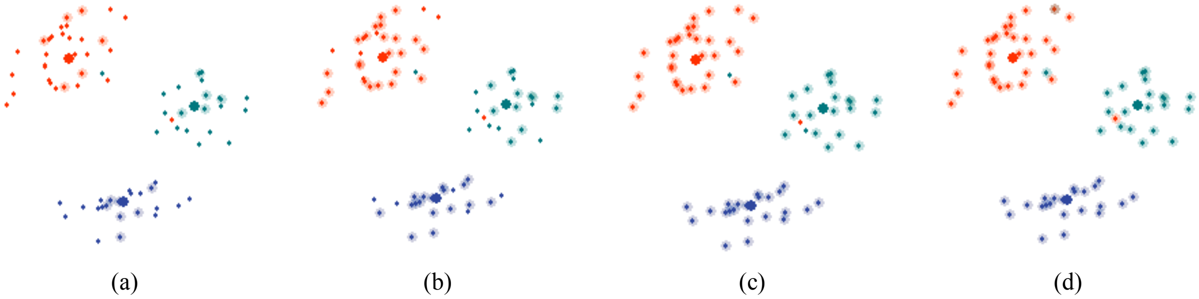
Acknowledgements

We wish to thank the other members of the Orca development team including Peter Sutherland, Tony Rossini, Thomas Lumley, and Nicholas Lewin-Koh for producing such a wonderful data visualization toolkit and for all the help they provided in using it. We also would like to thank Iowa State University for providing the funding through a Research Initiation Grant for work on this project.



(a) (b) (c) (d)

Figure 9. Convex hull visualization method with decreasing threshold values. (a) *t* = 0.87. (b) *t* = 0.75. (c) *t* = 0.44. (d) *t* = 0.38.



(a) (b) (c) (d)

Figure 10. Individual points visualization method with decreasing threshold values. (a) *t* = 0.87. (b) *t* = 0.75. (c) *t* = 0.44. (d) *t* = 0.38.

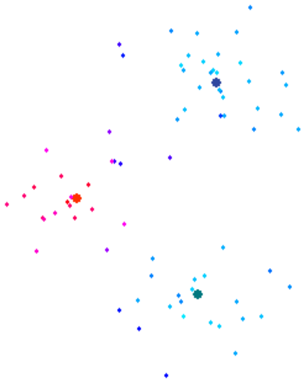


Figure 11. Color visualization method using temperature interpolation scheme.

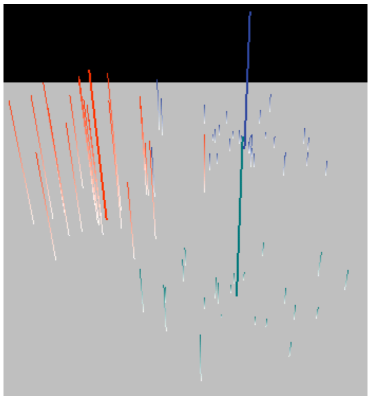
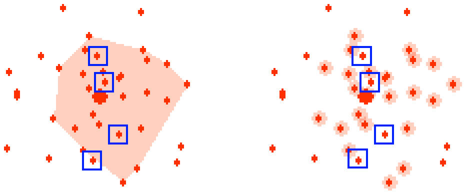


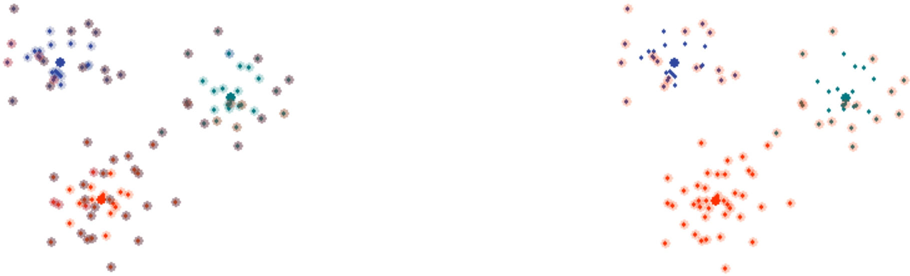
Figure 12. 3-D plot visualization method.



(a)

(b)

Figure 13. Illustration of data points with low membership values projected into convex hull. (a) Convex hull method. (b) Individual points method.



(a)

(b)

Figure 14. (a) Individual points method with all clusters produces a muddy plot. (b) Individual points method using only the red cluster produces a more understandable plot.

VISUAL EXPLORATION OF SELF-ORGANIZING MAPS USING THE GRAND TOUR AND LINKED BRUSHING

A paper to be submitted to the Journal of Statistical Software

Zach Cox

Abstract

The Self-Organizing Map (SOM) is a popular and well-studied unsupervised machine learning technique. Much work has been done recently on visualizing the results of the SOM, using static non-interactive approaches. This paper presents two new SOM visualization methods, based on the grand tour and linked brushing. These new methods use animation to show the SOM in the true high-dimensional space of the data, and interactivity to let the user explore the link between the map in input space and feature space. The new methods were implemented in Java using JSOMap, a new Java-based SOM package, and Orca, a Java-based data visualization package. Several different data sets are used to demonstrate the new methods.

Introduction

The Self-Organizing Map (SOM) is a very flexible machine learning tool. It is formulated mathematically as an unsupervised neural network, and during training it can be thought of as a non-parametric regression method to fit a low dimensional map to a data set's distribution in higher dimensions in an ordered fashion. After training, the SOM can be used for non-linear projection, mapping points in high dimensional input space onto a discrete set of points in a low dimensional feature space.

Researchers have used the SOM in numerous areas. (Kohonen 2001) claims over 4000 research papers have been written about SOM since its inception in the early 1980's and lists applications in machine vision, speech analysis, telecommunications, robotics, neurophysiology, chemistry, finance, and many others.

Research is now being done on how best to visualize the results of SOM and how to interpret these results. For example, (Kaski, Venna et al. 1999) color nodes of the map to accentuate the cluster structure of the data set, (Vesanto 1999) colors nodes of the map based on their models' values of single variables, and (Himberg 1998) links coloring between the map in feature space and Sammon's projection of the map in input space. These visualizations represent the end-results of the SOM algorithm and are all static; that is, the appearance of the nodes and models is fixed.

In an exploratory setting however, more insight can be gained about the SOM by using dynamic graphics techniques. The grand tour (Asimov 1985; Buja, Cook et al. 1997) allows the SOM to be visualized in the true high-dimensional space of the data by creating projections from the input space to 2-D. The grand tour animation also allows the iterations of the SOM algorithm to be visualized, showing exactly how the model vectors of the nodes in the map are updated as the algorithm progresses.

Allowing the user to interact with several different plots of the map provides a greater understanding of the outcome of the SOM algorithm. Linked brushing (McDonald 1982; Becker and Cleveland 1987) between a plot of the map in input space and a plot of the map in feature space lets the user identify certain nodes and models in the two views and interactively project data points onto the map.

Orca (Sutherland, Rossini et al. 2000), a Java-based data visualization package, was used for the creation of these new visualizations. Orca features excellent support for basic data visualization tasks such as appropriate data structures, file I/O, and user interfaces, allowing researchers to focus their efforts on processing and custom visualizations. In addition, JSOMap was created to handle all of the SOM-related computations. JSOMap (Cox 2002) is a flexible, extensible, object-oriented Java-based package for using Self-Organizing Maps.

Self-Organizing Maps

The SOM (Kohonen 2001) is an unsupervised neural network and can be used for non-parametric regression, non-linear projection, dimensionality reduction, feature extraction, clustering, and visualization depending on the interpretation and use of its results. Figure 15 below shows a typical architecture of the neural network.

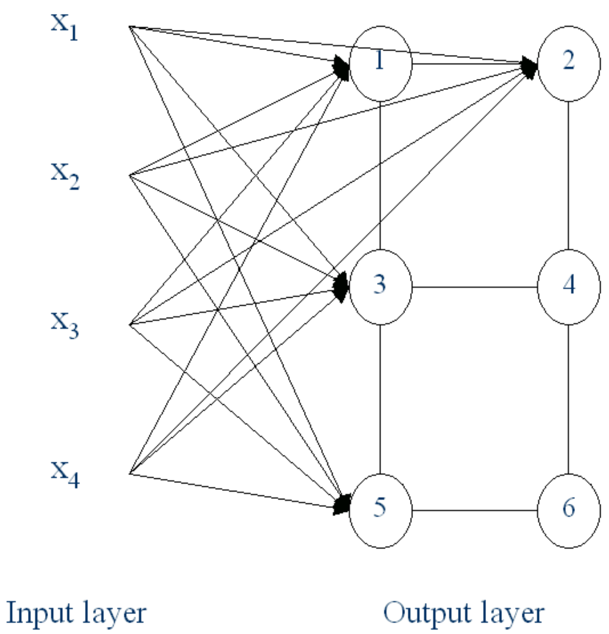


Figure 15. The SOM as an unsupervised neural network. The SOM consists of an input layer, where the input patterns are presented, and an output layer of connected neurons. These neurons exist as discrete points in some space, in this case 2-D, and each neuron has a weight vector of the same dimensionality as the input patterns.

Each neuron, or *node*, in the output layer has two vectors associated with it that exist in two different spaces: feature space and input space. Feature space is the space in which the nodes exist (which is typically 1-D or 2-D) and input space is the space in which the data exists (typically 2-D or higher). The first vector describes the position of the node in feature space, and the second represents the weight vector of the neuron. Each neuron has a weight associated with each input variable; the lines in Figure 15 represent these weights, and together they form a vector of the same dimensionality as the input data. These weight vectors can also be represented as points in input space; in this case, they are referred to as *models* since they are the input space representation of the nodes. When referring to the map as a neural network, it is natural to refer to neurons and weight vectors; when referring to points in feature space and input space, nodes and models are used instead.

The goal of the SOM algorithm is to adjust the weight vectors of the neurons to match the training data. One can also think of this as positioning the models of the nodes in input space to match the distribution of the data. The algorithm attempts to do this so that nodes near each other in feature space will have models near each other in input space. Positioning the models to approximate the data distribution can be thought of as *non-parametric regression*, much like discretized principal curves (Hastie and Stuetzle 1989; Mulier and Cherkassky 1995). The trained map can then perform *non-linear projection* of data points in input space, by finding the nearest model to the data point and projecting the point onto the model's node in feature space. For a more detailed description of the SOM algorithm, please see (Kohonen 2001)

The JSOMap Package

JSOMap (Cox 2002) is a Java-based package for working with Self-Organizing Maps. Utilizing object-oriented and smart software design principles (Gamma, Helm et al. 1995; Bloch 2001), it is intended to be incredibly easy to use in a variety of applications, from pure number crunching to interactive demos, while also being very flexible and supportive of customization. A collection of Java interfaces specifies the different parts involved in the SOM algorithm. Fully functioning default implementation classes, ready for use by the user, back these interfaces. Custom implementations are easily integrated into the existing class structure since the API refers to interfaces, not concrete classes.

There are three main parts to the SOM: the data, the map, and the algorithm. The data set is made up of patterns (or data points, cases, examples, etc.). The map is comprised of nodes and models, each of which represents a pattern in feature space or input space, respectively. The algorithm iteratively places the models of the map in input space to approximate the distribution of the patterns, and uses several other functions such as a distance metric and kernel.

Accordingly, a collection of interfaces represents these parts of the SOM in the JSOMap API. Figure 16 below shows a UML diagram of these main interfaces. The Pattern interface represents a basic pattern (or data point, case, example, etc.) and the Data interface is an indexed collection of Pattern objects. The Model and Node interfaces represent the models and nodes of the map, which are stored in an instance of the Map interface.

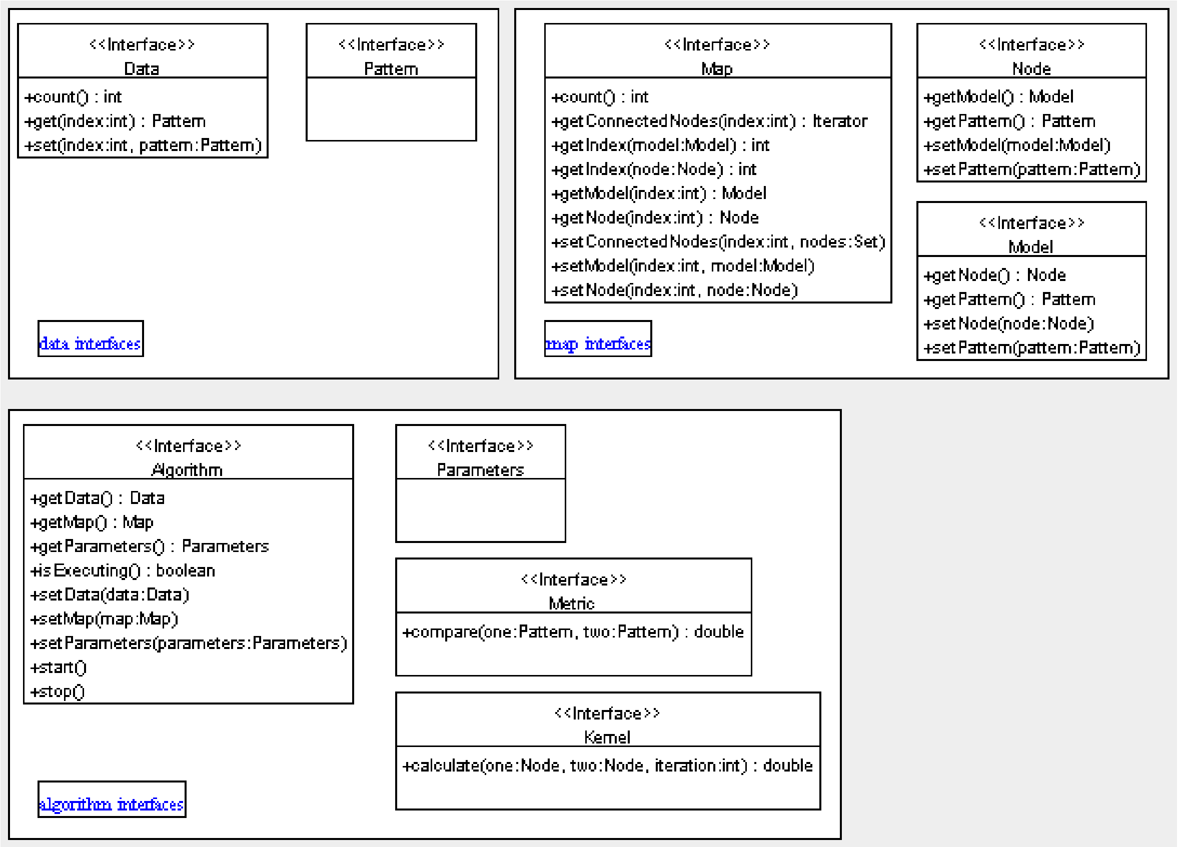

<<Interface>>
Data
+count() : int
+get(index:int) : Pattern
+set(index:int, pattern:Pattern)
<<Interface>>
Pattern
data interfaces
<<Interface>>
Map
+count() : int
+getConnectedNodes(index:int) : Iterator
+getIndex(model:Model) : int
+getIndex(node:Node) : int
+getModel(index:int) : Model
+getNode(index:int) : Node
+setConnectedNodes(index:int, nodes:Set)
+setModel(index:int, model:Model)
+setNode(index:int, node:Node)
<<Interface>>
Node
+getModel() : Model
+getPattern() : Pattern
+setModel(model:Model)
+setPattern(pattern:Pattern)
<<Interface>>
Model
+getNode() : Node
+getPattern() : Pattern
+setNode(node:Node)
+setPattern(pattern:Pattern)
map interfaces
<<Interface>>
Algorithm
+getData() : Data
+getMap() : Map
+getParameters() : Parameters
+isExecuting() : boolean
+setData(data:Data)
+setMap(map:Map)
+setParameters(parameters:Parameters)
+start()
+stop()
<<Interface>>
Parameters
<<Interface>>
Metric
+compare(one:Pattern, two:Pattern) : double
<<Interface>>
Kernel
+calculate(one:Node, two:Node, iteration:int) : double
algorithm interfaces

Figure 16. UML diagram of main interfaces of JSOMap API.

Many concrete implementations of these classes, performing basic functionality, are also provided in the JSOMap package. For instance, the following code segment shows a complete program to read the training data, initial models, node positions, and node adjacency from files, train the map, and print out the resulting models:

```
import java.io.*;
import jsomap.*;
import jsomap.data.*;
import jsomap.algorithm.*;
import jsomap.algorithm.parameters.*;

public class jsomapTest {

        public static void main(String args[]) {
                //read in data and create the map
                Data data = new DoubleDataReader(new FileReader("data.txt"));
                DoubleData nodes = new DoubleDataReader(new FileReader("nodes.txt"));
                DoubleData models = new DoubleDataReader(new FileReader("models.txt"));
                IntegerData adjMatrx = new IntegerDataReader(new FileReader("adjMatrix.txt"));
                Map map = new TraditionalMap(nodes, models, adjMatrix);

                //setup the parameters
                int iterations = 20;
                Metric metric = new EuclideanDoubleMetric();
                Kernel kernel = new GaussianKernel(metric, new Decay1(0.9d, 0.0001d,
                iterations));
                BatchUpdater updater = new BatchDoubleUpdater();
                Parameters params = new BatchParameters(iterations, metric, kernel, updater);

                //run the batch SOM algorithm
                Algorithm algorithm = new BatchAlgorithm(data, map, params);
                algorithm.start();

                //print the models
                System.out.println("models = ");
                for (int i=0; i<map.count(); i++) {
                    System.out.println("" + i + " = " + map.getModel(i).getPattern());
                }
            }
}
```

Specifying these components as interfaces instead of classes creates a very flexible API. For instance, in the above code example a `DoubleDataReader` instance is created, which implements the `Data` interface, and is given to the `Algorithm`. Any other concrete `Data` implementation could have been used to provide training data to the algorithm. For example, `BasicDoubleData`, which just wraps a `double[][]`, could have also been used. Or, a user could define a custom `Data` implementation, and give that to the algorithm as training data. This custom implementation could access any data source, such as a database or some existing data object.

JSOMap also provides support for event notification (via the `AlgorithmListener` interface and `AlgorithmEvent` class), multi-threaded use (via the `RunnableAlgorithm` wrapper class), and more fine-controlled algorithms (via the `ControllableAlgorithm` interface). A complete software design description is beyond the scope of this paper, although the above descriptions give a flavor of the overall API design. For more information, please see (Cox 2002).

Implementation in Orca

Orca is a toolkit for producing views of data (Sutherland, Rossini et al. 2000) using dynamic graphics. It is written entirely in Java and provides basic functionality for data visualization such as data file parsing, data processing, familiar plots, and user interface management. The software is structured by a pipeline architecture, in which the different pipe sections perform different aspects of data processing and rendering.

For example, a typical pipeline starts with a data source pipe, which is responsible for reading the data from a file into an internal array. This raw data (which is assumed in this project to be in arbitrary *p*-dimensions) is then passed to a standardization pipe where each variable is standardized, for instance to the range [-1, 1] or [0, 1]. The *p*-dimensional data may now be projected to a lower *d*-dimension for display ($d = 2$ in this project). The *d*-dimensional data is then converted to display coordinates and rendered for the user in a specific plot. The aforementioned pipeline is a basic, typical setup completely provided by core Orca classes.

This core package is also built heavily on Java interfaces, allowing Orca to be extended easily by end-users to create their own data processing and custom renderings. By providing a few specified methods these custom classes plug-in to the basic API, and gain access to the internal data structures of Orca. This saves an immense amount of development time, as the researcher can focus on specific experimental ideas instead of the rudimentary tasks done by the existing Orca pipe sections. Orca has been used to analyze multivariate time and space measurements in relation to studying the El Nino effect from buoys in the Pacific Ocean (custom time and spatial views of the data were created). It has also been used for research on interaction with graph data, where the nodes and edges have multiple variables.

Orca provides a grand tour (Asimov 1985; Buja, Cook et al. 1997) engine for projecting high-dimensional data to 2-D for display. The grand tour creates 2-dimensional projections of the *p*-dimensional data, and rotates smoothly between the projections. The smooth rotation creates a tour, or movie, of the data, which is very useful in determining the structure and features of a multivariate data set.

Orca also offers a basic scatter plot for rendering the data projected by the grand tour. However, this needed to be extended for rendering the models and connecting lines. Also, a custom scatter plot was needed for rendering the nodes in feature space. Figure 17 below shows the resulting Orca pipeline for the SOM visualization project. Pipe sections shaded gray are custom sections created for this project, while un-shaded sections are provided by Orca. Using the basic Orca pipe sections saved a good deal of time.

The above pipeline creates the two Orca plot windows shown in Figure 18 below. The main plot shows the grand tour view of the data points and models in input space. The user interface on this window controls execution of the algorithm, the map topology, and configures the brushing used on the plot. The second plot shows the nodes in feature space, and its user interface simply configures the brushing type.

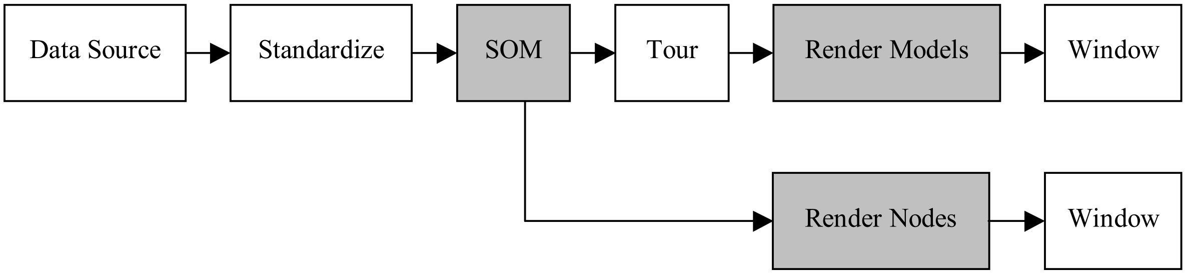
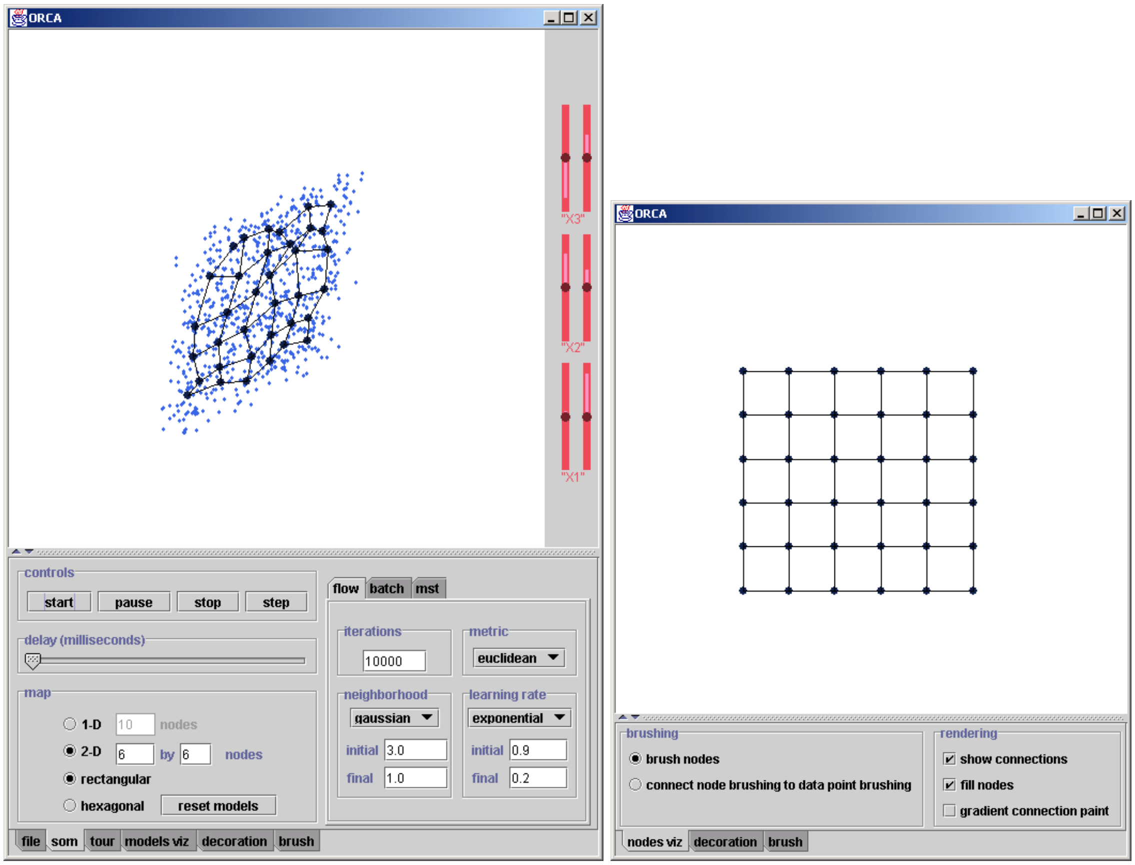


Figure 17. Orca pipeline for the SOM visualization project. Custom pipe sections are shaded gray.



controls

start | pause | stop | step

delay (milliseconds)

map

○ 1-D 10 nodes

● 2-D 6 by 6 nodes

● rectangular

○ hexagonal

reset models

flow | batch | mst

iterations 10000

metric euclidean

neighborhood gaussian

learning rate exponential

initial 3.0

final 1.0

initial 0.9

final 0.2

brushing

● brush nodes

○ connect node brushing to data point brushing

rendering

☑ show connections

☑ fill nodes

☐ gradient connection paint

Figure 18. The two Orca plot windows used to investigate the new SOM visualization methods. The plot on the left shows a snapshot of the grand tour, while the plot on the right shows the nodes in feature space.

Visualization of the SOM in Input Space

The first new SOM visualization method is an animation of the algorithm created in the grand tour view. This animation shows the models being positioned in the input space by the SOM algorithm. Figure 19 below shows an example, using a 3-D uniformly distributed data set, where the variance of the third variable is very small. The top left plot shows an early iteration, and the algorithm progresses to the end in the bottom right plot. Note that there are many iterations of the algorithm in between these snapshots.

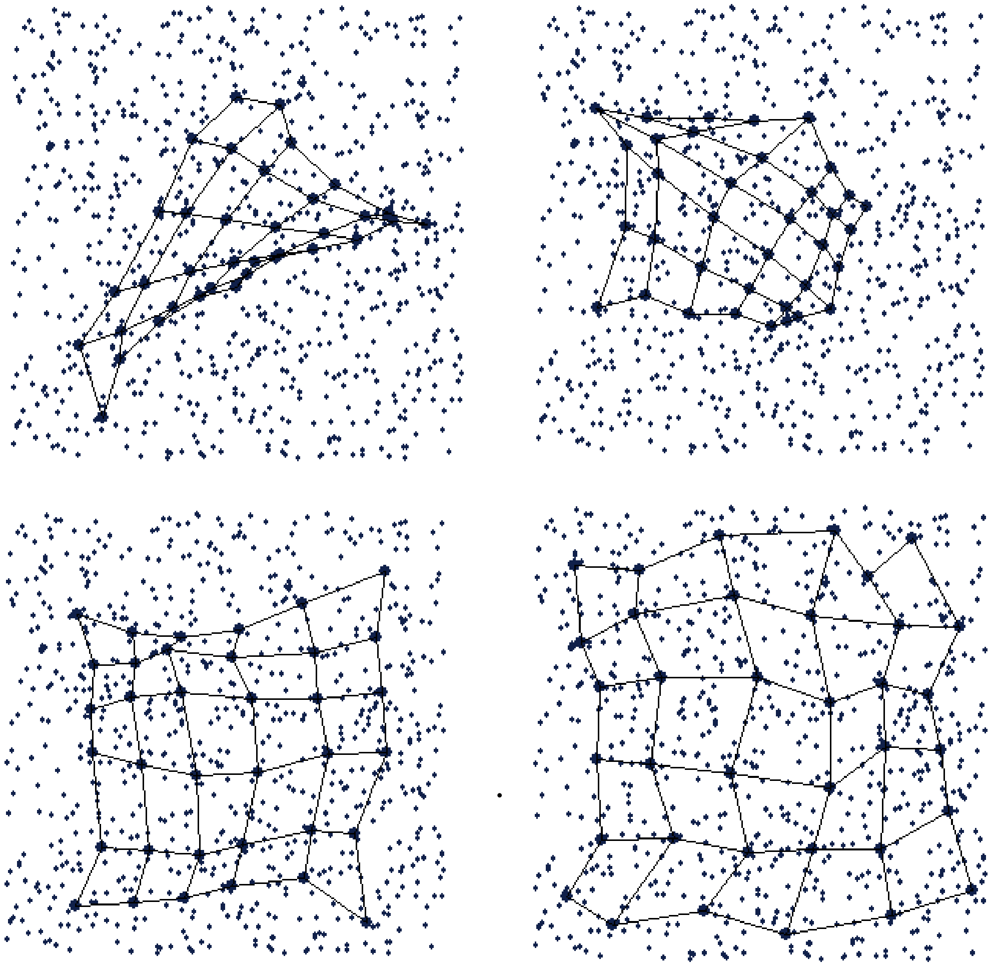


Figure 19. Successive iterations of the SOM algorithm, shown in an animation

The grand tour (Asimov 1985; Buja, Cook et al. 1997) was used as the method of choice for showing the data and models in input space for two main reasons. First, since the training data is multivariate and we want to view the algorithm positioning the models in input space with the data, some projection method is needed. The grand tour is a good method because it shows the data from different angles, allowing inherent structure of the data to be observed. Second, Orca provides convenient access to grand tour projections. Note that the grand tour is not the only possible projection method for visualizing the models and training data in input space; many other methods (PCA, CCA, projection pursuit, Sammon's mapping, etc.) could have also been used.

When viewed on-screen in real time, you can really see which model is being updated, and "pulling" the rest of the map along with it. This model corresponds to node *c* (see the Self-Organizing Maps section above), and the "elasticity" of the map is controlled by the neighborhood function, *n(c,i,k)*. Also, it is very informative to actually see the models being stretched around the data in input space. The animation therefore provides confidence in the proper functioning of the SOM algorithm. A static plot of the end-results of the algorithm may not provide this certainty.

The animation also provides a means for checking the parameters of the algorithm. For instance, almost all of the models should move towards the data point *x* during early iterations of the algorithm, while during later iterations only the models corresponding to node *c* and its immediate neighbors should move, and they should not move very far. If the neighborhood function is not performing this behavior, it can be easily identified during the animation and fixed. Watching the animation helped immensely during debugging of the JSOMap code.

Linked Brushing Between Grand Tour View and Map View

Linked brushing (McDonald 1982; Becker and Cleveland 1987) is a powerful graphical way of posing queries, or asking questions, about data. Multiple different views of the same data can be linked; a user can then query one view and have the other views provide the response to the query. For example, pair wise scatterplots of a multivariate data set can be graphically linked. The user can then brush data points in one plot to a new color and the same data points in the other plots will be updated to this color as well. In this case, the query is conditioning on the two variables in the first scatter plot. It basically means, "How do variables *x1* and *x2* affect the values of other variables in this data set?"

Linked brushing can also be used to pose queries about the SOM. By linking multiple views of the SOM, brushing on one view can ask a question which in turn is answered in another view. Three types of linked brushing are described in this paper:

- Brushing nodes and models
- Connecting data point brushing to node and model brushing
- Connecting node and model brushing to data point brushing

These brushing types provide answers to three different questions:

- Where did the algorithm position certain models in the input space?
- Which node(s) in feature space are certain data points projected to?
- Which data point(s) in input space are projected on to certain nodes in feature space?

Brushing Nodes and Models

Brushing nodes in the map view tells you where the models of those nodes are located in the tour view. This is related to the method in (Himberg 1998) except that it is interactive. The user can brush any node in the map view to any color, and the corresponding model in the tour view will be brushed in real time. Brushing can also be done on models in the tour view. In this case, brushing tells you the nodes in the map view that the brushed models belong to. This type of brushing can be used to gain more insight into the effects of the SOM algorithm.

Figure 20 shows an example of this type of brushing. The SOM algorithm positioned models of the map as shown in the tour view on the left. By brushing nodes in the map view on the right, it is easy to see the corresponding models in the tour view. Another realization is that the algorithm seems to have "flipped" the map. There is no requirement that the SOM algorithm must preserve any relative positions; just that neighboring nodes in feature space should have neighboring models in input space.

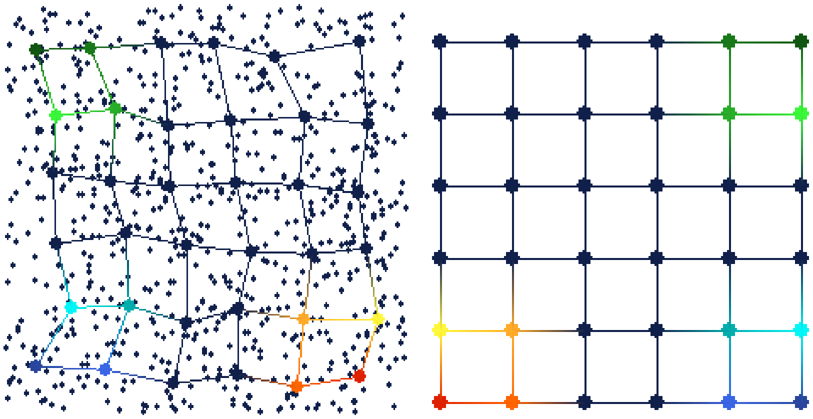


Figure 20. Brushing nodes in the map view (on the right) automatically brushes their corresponding models in the tour view (on the left). This type of brushing tells the user where the algorithm positioned the models in input space.

Connecting Data Point Brushing to Map Brushing

The second type of brushing tells you how data points in input space are projected onto the nodes of the map in feature space. The user can brush any group of points in the tour view, and the nodes they are projected on to are also brushed in the map view. For convenience, the corresponding models in the tour view are also automatically brushed.

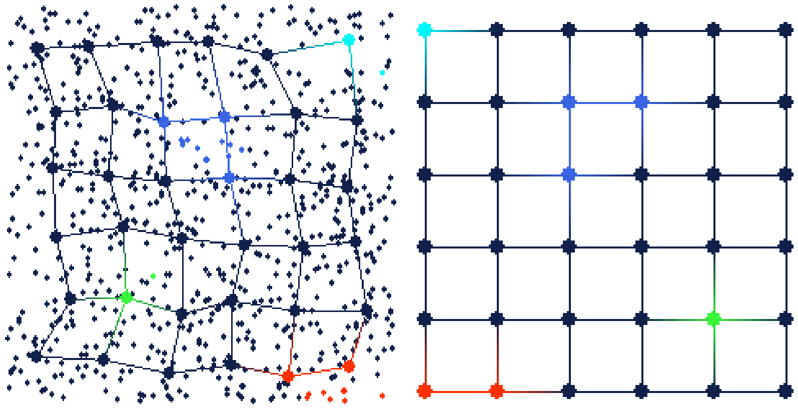


Figure 21. Brushing data points in the tour view (on the left) automatically brushes the nodes they are projected on to in the map view (on the right). This type of brushing tells the user how the data points are projected from input space to feature space.

Several data points in the tour view of Figure 21 have been brushed to different colors and projected on to the map view by linked brushing. This projection is done by simply finding the nearest model to the brushed data point in input space (using the Euclidean distance metric), and brushing the corresponding node in feature space. Thus, the user can interactively project data points from input space to feature space.

Connecting Map Brushing to Data Point Brushing

The third type of brushing tells you the data points in input space that are projected onto certain nodes in feature space. The user can brush nodes in the map view and all data points projected onto those brushed nodes are automatically brushed in the tour view. Models can also be brushed in this way in the tour view. In this case, all data points that are projected onto the nodes of those models are brushed in the tour view. The corresponding nodes in the map view are brushed as well.

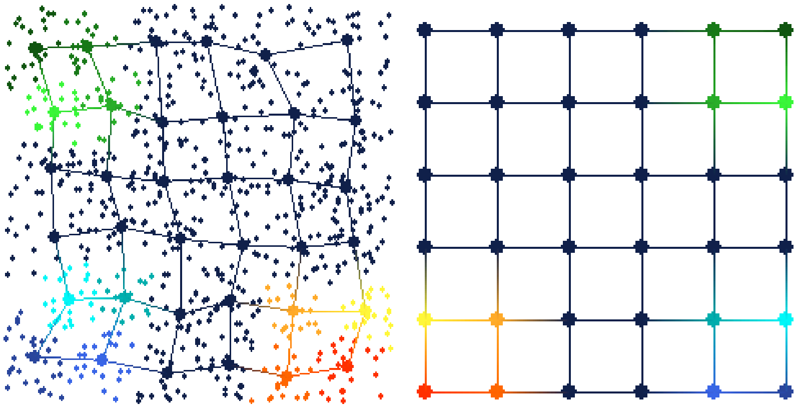


Figure 22. Brushing nodes in the map view (on the right) automatically brushes data points in the tour view (on the left) that are projected onto the brushed nodes. This type of brushing tells the user all of the data points in input space that are projected onto a specific node in feature space.

Figure 22 shows an example of this third type of brushing. Several nodes in the map view have been brushed with different colors (the same colors as in Figure 20). The data points that are projected onto these nodes are automatically brushed in the tour view. This type of brushing can roughly show the "projection boundaries" for the different nodes.

Applications

Several examples were presented in the previous section on linked brushing. The data set used in these examples is artificial; it is uniformly distributed in three dimensions, with a much smaller variance on the third variable. Since SOM is useful not only on toy data sets, an example with a real world data set is in order.

The data set presented here consists of various measurements on fleas. It consists of 75 data points (each representing a flea), 6 variables, and contains 3 classes of fleas. The classes of fleas are somewhat easily separated, by watching the grand tour. Each class tends to "move" across the screen in a different direction than the other classes during the tour. Figure 23 shows a snapshot of the tour, where the classes are visually separable.

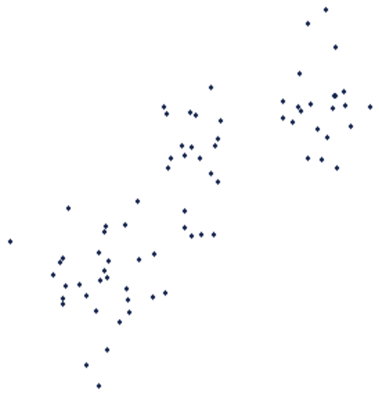


Figure 23. Grand tour projection of the flea data set. The three classes are visually separable in this projection.

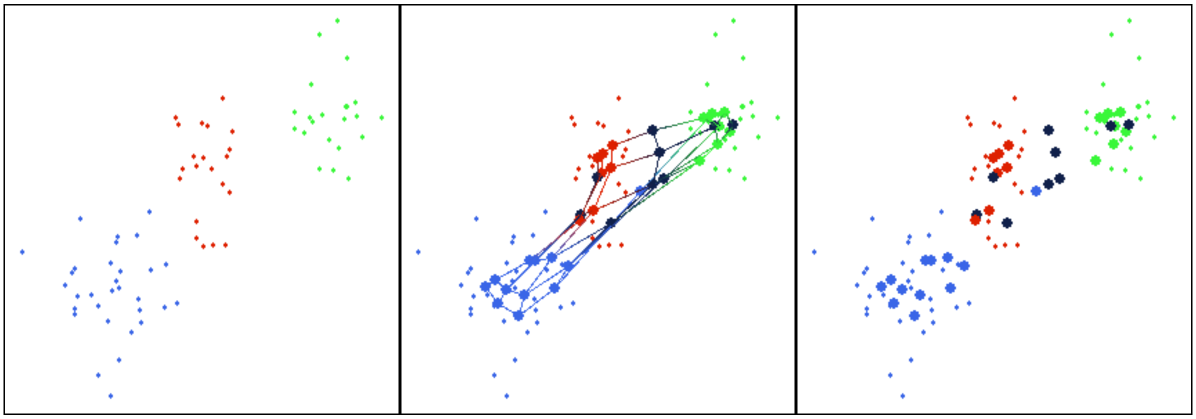


Figure 24. The data points in Figure 9 are brushed according to the visible classes (far left). The models are then added to the tour view, with adjacency connections (middle) and without (far right).

After fitting the SOM to this data, the data points in the tour view are brushed according to the classes seen in Figure 23. Figure 24 above shows the results of this brushing in the tour view. The data points are brushed without even seeing the models – the models are only added to the plot after the data points have been brushed.

You can see from the middle plot of Figure 24 that there are a much higher number of connections between models within the clusters than there are between models in different clusters. This suggests that the models within the clusters belong to adjacent nodes in the feature space. Naturally, this can be examined from the map view, shown below in Figure 25.

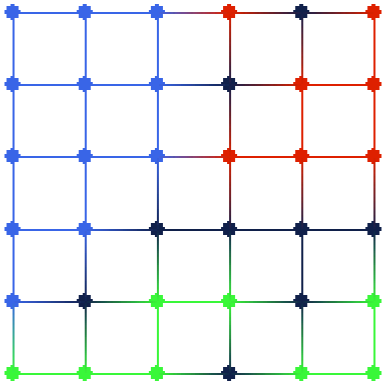


Figure 25. Map view after brushing in Figure 10. Note that the data points in the clusters in input space are projected to adjacent nodes in feature space. Also note the dark-colored nodes, which have no data points projected on to them.

It is very easy to see that data points in the same cluster in input space are projected on to adjacent nodes in feature space from the map view in Figure 25. Also, note that there are four nodes in Figure 25 that have no data points projected on to them, colored with dark blue. These nodes appear to be on the boundaries between the three clusters. Inspecting the tour view in Figure 24 again, it is apparent that these "empty" nodes do in fact lie between the clusters in input space.

By observing the grand tour, visually picking out clusters in the data, running the SOM algorithm, and brushing the data points based on their cluster, a sort of exploratory clustering has been performed. The SOM algorithm seems to agree with the clusters; remember that the algorithm positions the models of neighboring nodes in feature space to similar regions of input space.

When brushing the data points in the tour view, the following query is effectively posed: “Are these data points, which appear to be in the same clusters in input space, projected onto similar regions of feature space?” By observing the response in the map view, where the same colors appear in the same regions of the map and there are “empty” nodes between these regions, the answer is a definite “yes”.

This example shows that the SOM algorithm indeed positions the models in input space in an ordered fashion according to their corresponding nodes in feature space, and preserves distance relations among data points when projecting from input space to feature space. Also, the SOM algorithm has effectively done feature extraction on this data set. Originally, a supervised learning algorithm would have to create decision functions in 6-dimensional input space. However, after non-linear projection by the SOM, the supervised learning algorithm then only has to work in 2-D feature space, greatly simplifying the learning task. It is also obvious from the map view in Figure 25 that the projected classes are linearly separable in feature space.

Conclusions

This paper has presented two new methods for visualizing Self-Organizing Maps. The first, based on an animation of the SOM algorithm (or learning process) and the grand tour shows how the algorithm positions the models in input space to approximate the data’s distribution. The second method uses linked brushing between two different views of the SOM to allow a user to pose queries about the mapping between input space and feature space. Examples using two different data sets show the effectiveness of these two new visualization methods.

Although the tour view and map view are both good visual representations of the data and the SOM, they are definitely not the only possible informative plots. Many other ways of visualizing the data and models in input space are possible. These other views could also utilize the two new SOM visualization methods: they could show the animation of the SOM algorithm, and provide linked brushing between all of the views. Orca provides this linking capability, and the JSOMap package handles the SOM algorithm. All that is needed for new input space views are new Orca pipe sections that perform the appropriate projection from input space to 2-D and render the data and models to the screen.

CREATING METABOLIC NETWORK MODELS USING TEXT MINING AND EXPERT KNOWLEDGE

A chapter to be published in the book Computational Biology and Genome Informatics

J.A. Dickerson, D. Berleant, Z. Cox, W. Qi, D. Ashlock, E.S. Wurtele and A.W. Fulmer

Introduction

RNA profiling analysis and new techniques such as proteomics (the profiling of proteins) and metabolomics (the profiling of small molecules) are yielding vast amounts of data on gene expression. This points to the need to develop new methodologies to identify and analyze complex biological networks. This chapter describes the development of a Java™-based tool that helps dynamically find and visualize metabolic networks. The tool consists of three parts. The first part is a text-mining tool that pulls out potential metabolic relationships from the PubMed database. These relationships are then reviewed by a domain expert and added to an existing network model. The result is visualized using an interactive graph display module. The basic metabolic or regulatory flow in the network is modeled using fuzzy cognitive maps. Causal connections are pulled out from sequence data using a genetic algorithm-based logical proposition generator that searches for temporal patterns in microarray data. Examples from the regulatory and metabolic network for the plant hormone gibberellin show how this tool operates.

The goal of this project is to develop a publicly available software suite called the Gene Expression Toolkit (GET). This toolkit will aid in the analysis and comparison of large microarray, proteomics and metabolomics data sets. It also aids in the synthesis of the new test results into the existing body of knowledge on metabolism. The user can select parameters for comparison such as species, experimental conditions, and developmental stage. The key tools in the Gene Expression Toolkit are:

- **PathBinder: Automatic document processing system** that mines online literature and extracts candidate relationships from publication abstracts.

- **ChipView: Explanatory models** synthesized by clustering techniques together with a genetic algorithm-based data-mining tool.
- **FCModeler: Predictive models** summarize known metabolic relationships in fuzzy cognitive maps (FCMs).

Figure 26 shows the relationship between the different modules. The PathBinder citations are available to the researcher and smoothly transferable for use in annotating displays in other parts of the package and as links in building models. ChipView searches for link hypotheses in microarray data. The FCModeler tool for gene regulatory and metabolic networks is intended to easily capture the intuitions of biologists and help test hypotheses along with providing a modeling framework for putting the results of large microarray studies in context.

Structure of Concepts and Links

The nodes in the metabolic network represent specific biochemicals such as proteins, RNA, and small molecules(metabolites), or stimuli, such as light, heat, or nutrients. Three basic types of directed links are specified: conversion, regulatory, and catalytic. In a conversion link (black arrow, shown as a heavy dotted line), a node (usually representing a chemical) is converted into another node, and used up in the process. In a regulatory link (green and red arrows, shown as solid and dashed arrows respectively), the node activates or deactivates another node, and is not used up in the process.

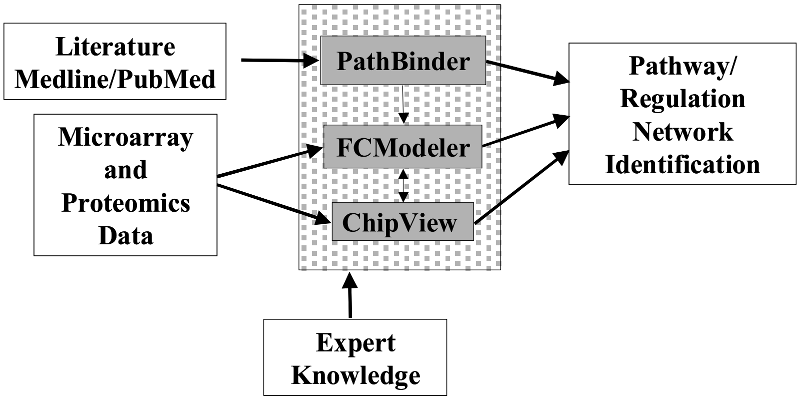


Figure 26. The Gene Expression Toolkit consists of PathBinder, FCModeler, and ChipView. The inputs to the system are the literature databases such as PubMed; experimental results form RNA microarray experiments, proteomics, and the expert knowledge and experience of the biologists that study an organism. The result will be a predictive model of the metabolic pathways.

A catalytic link (blue arrows, shown as a thick line) represents an enzyme that enables a chemical conversion and does not get used up in the process. Figure 27 shows a small part of a graph for the Arabidopsis metabolic and regulatory network. There is also an undirected link that defines a connection between two nodes and does not specify a direction of causality.

In the metabolic network database, the type of link is further delineated by the link mechanism and the certainty. Some of the current mechanisms are: direct, indirect, and ligand. Direct links assume a direct physical interaction. Indirect links assume that the upstream node activates the downstream node indirectly and allows for the existence of intermediate nodes in such a path. The ligand link is a “second messenger” mechanism in which a node produces or helps produce a ligand (small molecule that binds) and either “activates” or “inhibits” a target node. Often the nature of the link is unknown and it cannot be modeled in the current framework. The link certainty expresses a degree of confidence about the link. This will be used for hypothesis testing.

Other key features include concentrations of the molecules (nodes), strengths of the links, and subcellular compartmentation. These data can be added as they are identified experimentally. Currently the biologist user can include or ignore a variety of parameters, such as subcellular compartmentation and link strength. Since the node and link data is entered into a relational database, individual biologists can easily sort, share, and post data on the web. Future versions will distinguish between regulation that results in changes in concentrations of the regulated molecule, and regulation that involves a reversible activation or deactivation.

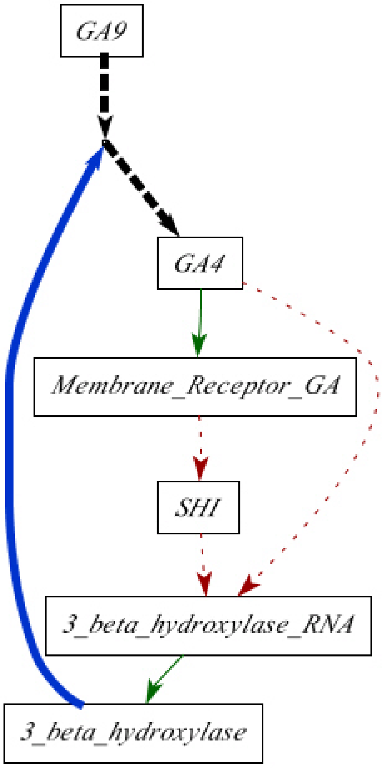


Figure 27. This is a map of a simple metabolic model of gibberellin (active form is GA4). The sequence is started by translation of 3_beta_ hydroxylase_RNA into the 3_beta_ hydroxylase protein. Bold dashed lines are conversion links, bold lines are catalytic links, thin solid lines are positive regulatory links and dashed thin lines are negative regulatory links.

PathBinder: Document Processing Tool

PathBinder identifies information about the pathways that mediate biological processes from the scientific literature. This tool searches through documents in MEDLINE for passages containing terms that indicate relevance to signal transduction or metabolic pathways of interest. Microarray data can be used to hypothesize causal relationships between genes. PathBinder then mines MEDLINE for information about these putative pathways, extracting passages most likely to be relevant to a particular pathway and storing this desired information. The information is presented in a user-friendly format that supports efficiently investigating the pathways.

Related Work on Knowledge Extraction from Biochemistry Literature

An increasing body of work addresses extraction of knowledge from biochemical literature. Some works compare documents, such as MEDLINE abstracts, and extract information from the comparisons. For example, Shatkay et al. and Stapley assess the relatedness of genes based on the relatedness of texts in which they are mentioned (Shatkay 2000; Stapley 2000). Shatkay et al. get documents containing a particular gene, compare the set of documents to the set relevant to other genes, and if two sets are similar then the two genes are deemed related. Stapley compares the literatures of two genes and assesses relatedness of genes based on the rate at which papers contain both of them. The system presented by Usuzaka et al. learns to retrieve relevant abstracts from MEDLINE based on examples of known relevant articles (Usuzaka 1998).

Other works directly address the relationships among entities such as proteins, genes, drugs, and diseases. An initial requirement for such a system is identifying relevant nouns. This can be done by extracting names from free text based on their morphological properties. Sekimizu et al. [1998] parse text to identify noun phrases, rather than concentrating on the nouns themselves. The GENIA system and the PROPER system address the need to identify relevant terms automatically to enable automatic maintenance of lexicons of proteins and genes (Fukuda, T. Tsunoda et al. 1998; Collier 1999). Proux et al. [1998] concentrate on gene names and symbols.

Once the lexicon problem has been addressed, text can be analyzed to extract relationships among entities discussed therein. (Andrade and Valencia 1998) extract sentences that contain information about protein function. Rindflesch et al. [1999] concentrate specifically on binding relationships (among macromolecules). Rindflesch et al. [2000] emphasizes drug-gene-cell relationships bearing on cancer therapy. Thomas et al. [2000] use automatic protein name identification to support automatic extraction of interactions among proteins. Sekimizu et al. [1998] use automatically identified relevant noun phrases in conjunction with a hand-generated list of verbs to automatically identify subject-verb-object relationships stated in texts in MEDLINE. Craven and Kumlien [1999] extract relationships between proteins and drugs. They investigate two machine learning techniques in which a hand-classified training set is given to the system, which uses this set to infer criteria for deciding if other passages describe the relevant relationships. One machine learning technique is based on modeling passages as unordered sets of words, and assumes word co-occurrence probabilities are independent of one another (the Naïve Bayes approach). Tanabe et al. [1999] extract relationships between genes and between genes and drugs. Their MedMiner system supports human literature searches by retrieving and serving sentences from abstracts on MEDLINE over the Web, based on their keyword content. MedMiner is tuned to finding relationship-relevant sentences in abstracts that contain a gene name and relationship keyword, pair of gene names and relationship keyword, or a gene and a drug name and relationship keyword. MedMiner can also handle Boolean queries, such as those containing two protein names. In such cases MedMiner takes a query consisting of an OR'ed list of "primary" terms and an AND'ed list of "secondary" terms. A returned sentence must contain a "primary" term and a relationship word. Relationship words are from a relatively large lexicon of such terms predefined by the system.

A number of works address extracting relationships among proteins from biochemical texts. A solution enables both automatic construction of biochemical pathways, and assistance to investigators in identifying relevant information about proteins of interest to them.

Humphreys et al. [2000] specifically address enzyme reactions extracted from *Biochimica et Biophysica Acta* and *FEMS Microbiology Letters*. Such interactions are intended to support metabolic network construction. Rindflesch et al. [1999] apply non-trivial natural language processing (NLP) to extract assertions about binding relationships among proteins. Noun phrases are identified by a sophisticated combination of text processing and reference to existing name repositories.

Other systems have been reported that extract many interactions among diverse proteins. Blaschke et al. [1999] extracts such interactions by first identifying phrases conforming to the template `protein... verbclass ...protein`, where `verbclass` is one of 14 sets of pathway relevant verbs (such as "bind") and their inflections. Protein names and synonyms are provided as an input and sentences containing extracted phrases are returned. The BioNLP subsystem, a component of a larger system, extracts sentences containing pathway relevant verbs determined by the user and applies templates to them to identify path relevant relationships among proteins (Ng 1999; Wong 2001). Protein names are determined automatically. The subsystem, CPL2Perl, thresholds the results so that it ignores interactions with a single relevant sentence. This is useful if the sentence analysis was mistaken. Such a thresholding strategy tends to increase precision at the expense of reducing recall. Thomas et al. [2000] distinguish between verbs that are relatively more and less reliable in indicating protein interactions. Their system automatically recognizes protein names and relies on the strategy of tuning an existing sophisticated general-purpose natural language processing system to the protein interaction domain. Ono et al. [2001] use part-of-speech (POS) tagging, key verbs, and template matching on phrases to extract protein-protein interactions. Their system has an information retrieval effectiveness measure of up to 0.89 (Ding, Berleant et al. 2002).

PathBinder Operation

The PathBinder system, like previous works, extracts relevant passages about protein relationships from MEDLINE. The PathBinder work differs from these due to a combination of system design decisions. PathBinder avoids syntactic analysis of text in favor of word experts for pathway relevant verbs. Word experts are sets of rules for interpreting words (Berleant 1995). PathBinder also is oriented toward assisting humans in constructing pathways rather than fully automatic construction, thus avoiding some information retrieval precision limitations. We are also investigating the relative performances of several algorithms for identifying relevant sentences, including verb-free algorithms that rely instead on protein term co-occurrences. PathBinder relies on the sentence unit rather than abstracts, phrases, or other units because sentences rate highly on information retrieval effectiveness under reasonable conditions (Ding, Berleant et al. 2002).

How PathBinder Works

Step 1: user input. Keyboard input of biomolecule names in pathways of interest by the user.

Step 2: synonym extraction. A user-editable synonym file is combined with a more advanced module that will automatically access the HUGO (http://www.gene.ucl.ac.uk.publicfiles/nomen/nomenclature.txt) and OMIM (www.ncbi.nlm.nih.gov/htbinpost/Omim/) nomenclature databases, and extract synonyms.

Step 3: document retrieval. PubMed is accessed and queried using terms input in Step 1. The output of this step is a list of URLs with high relevance probabilities.

Step 4: sentence extraction. Each URL is downloaded and scanned for pathway-relevant sentences that satisfy the query. These sentences constitute pathway-relevant information “nuggets.”

Repetition of steps 2 through 4, using different biomolecule names extracted from qualifying sentences. These new biomolecule names are candidates for inclusion in the pathways of interest.

Step 5: sentence index. Process the collection of qualifying sentences into a more user-friendly form, a multi-level index (Figure 28), with the number of levels dependent on the sentence extraction criteria. This index conforms to a pattern, displayed by a Web browser, and the sentences in it are clickable. When a sentence is clicked, the document from which it came appears in the Web browser.

Step 6: integration with the rest of the software and the microarray data sets. The index can be used to create a graphical representation in which verbs are represented by lines, interconnecting the biomolecule names and forming a web-like relationship diagram of the extracted information.

PathBinder is useful as both a standalone tool and an integrated subsystem of the complete system. The multilevel indexes transform naturally into inputs for the network modeling tools. The networks that PathBinder helps identify will form valuable input to the clustering, display, and analysis software modules.

```
Protein A
    Protein B
        Associates/Associated/etc.
            Sentence 1
            Sentence 2
            . . .
        Binds/Binding/Bind/etc.
            Sentence M
            Sentence M+1
            . . .
        Regulates/Regulating/etc.
        . . .
    Protein C
        Associates/Associated/etc.
            Sentence M+N
            . . .
        Binds/Binding/Bind/etc.
        . . .
Protein B
    Protein D
        Associates/Associated/etc.
            Sentence M+N+P
```

Figure 28. The long and somewhat disorganized sentence set that PathBinder extracts is converted into a multilevel index which is more suited to a human user. "Protein A", "Protein B", etc. are placeholders for the actual name of a path-relevant protein, and "Sentence 1", "Sentence 2", etc. are placeholders that would be actual sentences in the PathBinder-generated index.

Example of a Sample PathBinder Query

The query is to find sentences containing (either gibberellin, gibberellins, or GA) AND (either SPY, SPY-4, SPY-5, or SPY-7). Three relevant results were found and incorporated into the metabolic and regulatory visualization. A single sentence example is show below.

Sentence: "The results of these experiments show that spy-7 and gar2-1 affect the GA dose-response relationship for a wide range of GA responses and suggest that all GA-regulated processes are controlled through a negatively acting GA-signaling pathway."

Source Information: UI—99214450, Peng J, Richards DE, Moritz T, Cano-Delgado A, Harberd NP, Plant Physiol 1999 Apr; 119(4):1199-1208.

ChipView: Logical Proposition Generator

Gene expression data is gathered as a series of snapshots of the expression levels of a large number of genes. The snapshots may be organized as a time series or a sequence of organism states. When multiple gene expression experiments are performed, the choice of genes, time points, or organism states often varies. Finally, the data gathered often contain many unusable points for a number of reasons. The variation in which data is collected, the noisy character of the data, and the fact that data is often missing mean that a gene expression analysis tool must be designed with all these limitations in mind. Current analysis tools, mostly built around clustering of various sorts, are quite valuable in cutting through the thickets of data generated by gene expression technology to find nuggets of truth (Eisen, Spellman et al. 1998; Brown, Grundy et al. 2000). These tools, however, do not currently suggest possible interpretations to the researcher and incorporate many ad hoc assumptions about the mathematical and algorithmic behavior of various clustering techniques.

One possible way of addressing both the data collection limitations and lack of theoretical foundation is the Logical Proposition Generator. The key features of this tool are:

- Filtration of data items by behavioral abstractions that yield both interpretation of data and partial resistance to variations in data collection.
- Incorporation of a vast space of clustering techniques into the tool to create data driven, problem-specific clustering on the fly.
- Designing the tool so that its basic data objects are logical propositions about the data it is working with.

This makes the analogy to clustering in the logical proposition generator one that transparently supplies multiple potential interpretations of the data. The output of the tool is in the form of logical sentences with atoms drawn from absolute and differential classifications of expression profiles and relative abstractions of pairs of gene expression profiles. The prototype tool was written for gene expression profiles that are time series. The goal is to extend the logical proposition generator to have logical primitives that are appropriate for non-time series data are one of the goals as well.

Operation of the Logical Proposition Generator

Let us now specify the atoms and connective of the logical proposition language that is the target of the tool's search of the data for meaning. The tool permits the user to specify the expression level *E* that they believe specifies up or down regulation of a gene and the minimum change in expression level *D* that represents a significant change between adjacent time points. The tool recognizes classes of expression profiles given by the regulation state at each time point. Thus, "up, not down, not unchanged, down, down, not up, unchanged," specifies one of the possible classes of a seven point time series. Likewise, if +/– means significant change up or down since the last time step "+++00 – –" would represent a class of profiles that first increased, then stayed level, and later decreased their regulation between time steps. These two types of classes of expression profiles form the single expression profile atoms of the language.

The tool also uses logical atoms that compare pairs of profiles. These compute representative facts about the profiles, such as "profile one has its maximum before profile two", "the maximum change in regulation of the second profile exceeds that of the first", or "upregulation in the first profile does not occur unless a change in regulation has occurred in the second". The absolute and differential (single expression profile) atoms and the relative (two expression profile) atoms both return a "true" or "false" result. With these atoms available we then use traditional Boolean connectives AND, OR, NOT, XOR, etc. to build logical propositions.

Once we have the ability to make logical statements about gene expression profiles, the problem them becomes locating interesting and informative propositions. Statements that are always true, tautologies, are not interesting. Instead, we use a form of evolutionary computation, genetic programming (Koza 1992; Kinnear 1994; Koza 1994; Angeline 1996) to locate propositions that are true of subsets of the expression profiles. While this can be done blindly, with utility similar to clustering, it is also possible to force the expressions to be true when one of their arguments comes from a restricted class of genes of interest, e.g. a class we are trying to modify the expression of by some intervention. Thus, to find genes important to the upregulation of a class of genes *X*, we would search for propositions $P[x, y]$ that are often true when *x* is in *X*, seldom true when *x* was not in *X*, for some substantial but **not** universal collection *Y* of values for *y*. These vague statements about "usually true" and "substantial" become mathematically precise when embedded into the evolutionary search tool as a fitness function. One target of the research is an understanding of which fitness function among those possible provide results useful to biological researchers.

The relation $\{x \in 2233333\} \wedge \{y \in 5566666\} \wedge \{x \textit{ first up before } y\}$ defines a binary relation of expression profiles. *x* must **not** change significantly at first while *y* must change at first. Later, *x* **must not go down** while *y* **must not go up** OR the first significant upregulation of *x* **must be before** that of *y*. Evolving such expressions permits the computation of interesting hypotheses about relations between profiles *including* relationships that use edges in the graphical models.

The logical proposition generator, by working with abstractions of the data in the form of the logical atoms described above yields the advantage that it is resistant, though certainly not immune, to variations in exactly which data are collected. The absolute and differential expression classes represent primitive fragments, which Boolean operations fuse together into data partitions, i.e. clusters. This means that the clustering techniques required to make sense of gene expression data are incorporated transparently into the logical proposition generator. Finally, in addition to locating genes that are implicated in the regulation of genes of interest, something clustering tools can do to some degree, the logical character of the tool will sometimes simultaneously suggests the "what" or "why" of the relationship, easing the work of interpretation and providing a source of tentative links for the other tools. This tool is not intended to replace clustering tools but to complement them. One way to locate a target set of genes, for example, might be to choose a tight cluster containing a few genes of interest and use this as a group of interest for the logical proposition generator.

Table 1. Codes for changes in the expression profiles.

Code	Measurement Change
1	Upregulated
2	Didn't change significantly
3	Didn't downregulate
4	Downregulated
5	Changed significantly from the baseline
6	Didn't upregulate
7	Matches anything

Example of Logical Proposition Generator Operation

The logical proposition generator operates on sets of expression profiles. It characterizes desired sequences as a series of numbers, e.g. Y in L: 124 means that Y is in the set of profiles that are in the state “Upregulated, didn’t change, and downregulated”.

Table 1 gives the codes used in this example. An example logical proposition is given below:

```
(NAND
        (NOR
                (Y in L:757243126155)
                (NAND (SamePro Y X) F))
                (AND T (NOT (NOT (NOR F T)))
        )
)
```

This is a logical proposition that acts on two 12-time-point expression profiles *X* and *Y*. It uses the logical operations *NAND*, *NOR*, *NOT*, and *AND* and the constants *T* and *F*. The logical proposition uses the binary predicate "*SamePro*" which is true if two profiles are significantly up-and-down regulated in the same pattern. It also uses the unary predicate "*Y* in *L*:525634163157" which tests to see if *Y* is in the class of profiles that displays a particular pattern of up and down regulation in its twelve time points according to the scheme in Table 1.

Logical propositions of this form have the potential to encode very complex classes of expression profiles in very short statements. The following logical proposition also uses *OR* and *Say*, which we use to encode the logical identity, as well as differential classes, e.g. "*X* in *D*:73512467452" which check for changes in regulation since the last time step rather than as compared to the baseline:

```
(NOR (Say (X in D:73512467452))
   (Say (OR (OR (X in D:71661716551) (X in L:177621456644))
      (NAND T (Say (Y in D:13376357161))))
   )
)
```

The *Say* operation does nothing but it leaves space in an expression that makes it easier for the evolutionary training techniques we use to move around sub-expressions that form coherent logical units.

Fuzzy Cognitive Map Modeling Tool for Metabolic Networks

The FCModeler tool for gene regulatory and metabolic networks captures the known metabolic information and expert knowledge of biologists in a graphical form. The node and link data for the metabolic map is stored in a relational database. This tool uses fuzzy methods for modeling network nodes and links and interprets the results using fuzzy cognitive maps (Kosko 1986; Kosko 1986; Dickerson and Kosko 1994). This tool concentrates on dynamic graphical visualizations that can be changed and updated by the user. This allows for hypothesis testing and experimentation.

Metabolic Network Mapping Projects

Two existing projects for metabolic networks are the Kyoto Encyclopedia of Genes and Genomes (Kanehisa and Goto 2000) (KEGG http://www.genome.ad.jp/kegg) and the WIT Project (Overbeek, Larsen et al. 2000) (http://wit.mcs.anl.gov/WIT2/WIT). The WIT Project produces “metabolic reconstructions” for sequenced (or partially sequenced) genomes. It currently provides a set of over 39 such reconstructions in varying states of completion from the Metabolic Pathway Database constructed by Evgeni Selkov and his team. A metabolic reconstruction is a model of the metabolism of the organism derived from sequence, biochemical, and phenotypic data. This work is a static presentation of the metabolism asserted for an organism. The purpose of KEGG is to computerize current knowledge of molecular and cellular biology in terms of the information pathways that consist of interacting genes or molecules and, second, to link individual components of the pathways with the gene catalogs being produced by the genome projects. These metabolic reconstructions form the necessary foundation for eventual simulations.

E-CELL is a model-building kit: a set of software tools that allows a user to specify a cell's genes, proteins, and other molecules, describe their individual interactions, and then compute how they work together as a system (Tomita, Hashimoto et al. 1997; Tomita, Hashimoto et al. 1999; Tomita 2001). Its goal is to allow investigators to conduct experiments “in silico.” Tomita's group has used versions of E-CELL to construct a hypothetical cell with 127 genes based on data from the WIT database. The E-CELL system allows a user to define a set of reaction rules for cellular metabolism. E-CELL simulates cell behavior by numerically integrating the differential equations described implicitly in these reaction rules.

EcoCyc is a pathway/genome database for *Escherichia coli* that describes its enzymes, and its transport proteins (Karp, Riley et al. 2000) (http://ecocyc.DoubleTwist.com/ecocyc/). MetaCyc is a metabolic-pathway database that describes pathways and enzymes for many different organisms. These functional databases are publicly available on the web. The databases combine information from a number of sources and provide function-based retrieval of DNA or protein sequences. Combining this information has aided in the search for effective new drugs (Karp, Krummenacker et al. 1999). EcoCyc has also made significant advances in visualizing metabolic pathways using stored layouts and linking data from microarray tests to the pathway layout (Karp, Krummenacker et al. 1999).

Visualizing Metabolic Networks

The known and unknown biological information in the metabolic network is visualized using a graph visualization tool. Figure 29 shows a screenshot of the FCModeler tool display window. The graph visualization is based on for visualizing and interacting with dynamic information spaces. FCModeler uses *Diva*, a Java-based software information visualization package (see http://www.gigascale.org/diva/) for its basic graph data structure, rendering, and interaction controls. In addition, it extends *Diva* to provide custom graphics-related features such as dynamic figures, graph layout, and panning and zooming. This allows for a greater variety of visualization objects on the display. The front end of the FCModeler tool is a Java^{TM} interface that reads and displays data from a database of links and nodes. The graph layout program is *dot*, which is part of the *Graphviz* program developed at AT&T research labs (see http://www.research.att.com/ sw/tools/graphviz/).

The nodes and edges in the FCModeler graph have properties, which can be specified in an XML file or created at run-time by the user. There is a set of properties for nodes and also one for edges. In a bioinformatics application, a node property may be “type of node”. Then each node would have a specific value for this property, such as “DNA”, “RNA”, “protein”, “environmental factor”, etc. Similarly, an edge property could be “type of reaction” with the specific values “conversion” or “regulatory.” Figure 30 shows the visual property window from FCModeler for some of the nodes and edges of the Arabidopsis graph shown in Figure 29.

Interaction

FCModeler currently supports several forms of user interaction with the graph model and view. One basic form of interaction is selection. Node and edge figures can be selected individually by clicking on them with the mouse, or by dragging a selection rectangle around a group of them. The selected node and edge figures are then visually distinguished from the rest

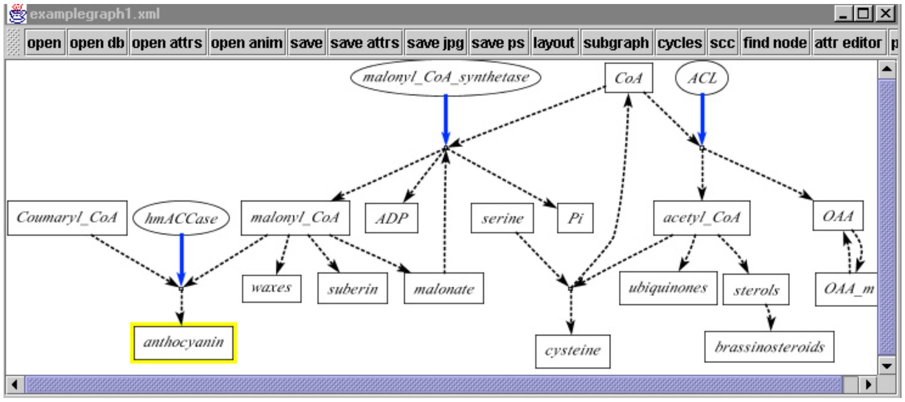
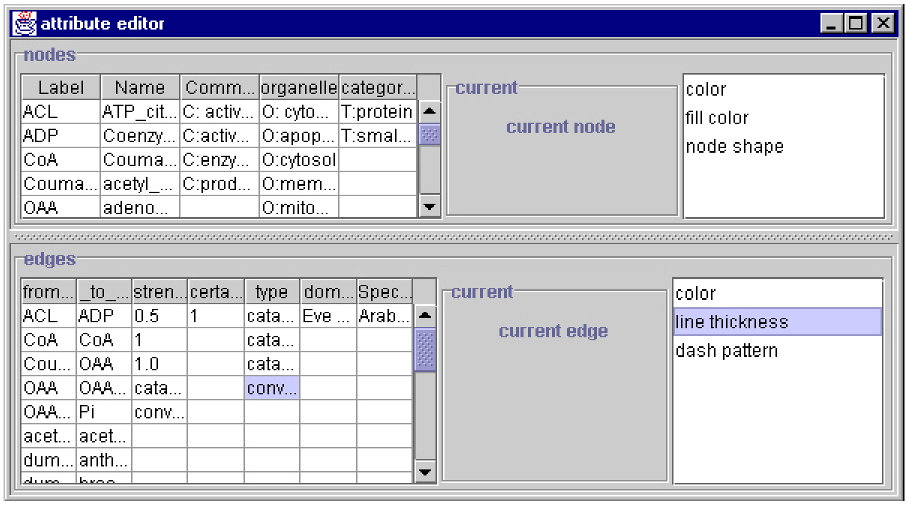


Figure 29. Screenshot of an FCModeler graph. The bold blue arrows represent catalyst links. The dashed arrows are conversion links. The proteins are shown as ellipses. The rectangles are small molecules. Nodes of interest can be highlighted by the user.



attribute editor

nodes

Label	Name	Comm...	organelle	categor...
ACL	ATP_cit...	C: activ...	O: cyto...	T:protein
ADP	Coenzy...	C:activ...	O:apop...	T:smal...
CoA	Couma...	C:enzy...	O:cytosol	
Couma...	acetyl_...	C:prod...	O:mem...	
OAA	adeno...		O:mito...	

current

current node

color
fill color
node shape

edges

from...	_to_...	stren...	certa...	type	dom...	Spec...
ACL	ADP	0.5	1	cata...	Eve ...	Arab...
CoA	CoA	1		cata...		
Cou...	OAA	1.0		cata...		
OAA	OAA...	cata...		conv...		
OAA...	Pi	conv...				
acet...	acet...					
dum...	anth...					

current

current edge

color
line thickness
dash pattern

Figure 30. The attribute editor in FCModeler. The color, shape, and fill of the nodes can be changed according to the existing properties. The color, line thickness, and dash pattern can be changed for the edges.

by some form of highlighting. Selection of node and edge figures can provide a starting point for other operations on the graph.

The user can reposition the nodes and edges on the screen by dragging them with the mouse. All of the selected figures will then be translated in the direction of the mouse movement. In addition, edge figures are rendered as Bezier curves (Angel 2000) and dragging with the mouse relocates the edge figures' individual control points.

FCModeler supports graphical modification the underlying metabolic map model. Node and edge figures can be added to and removed from the view. The user can also change the tail or head node of an edge by dragging the desired edge end to a new node figure.

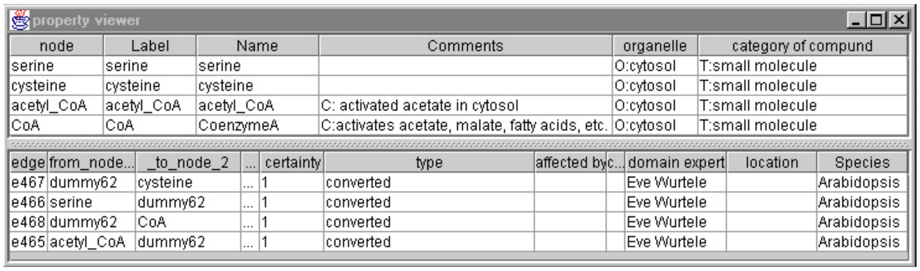
Zooming and panning allow the user to examine different parts of the graph in varying levels of detail. The graph may just be too large to be viewed as a whole on the screen, or a layout algorithm could use more space than is viewable at once for its layout. The view port can also be programmatically set to arbitrary coordinates.

Graph Layout

Any Diva graph view can use an arbitrary graph layout algorithm to compute the positions of its node and edge figures. Diva comes with several layout algorithms, but opens its views to custom implementations. FCModeler uses the Dot graph layout engine, which is part of the Graphviz graph drawing software from AT&T labs (http://www.research.att.com/sw/tools/graphviz/). Dot produces fairly nice layouts, and is easy to use. However, other more specialized layout algorithms may produce better layouts for the specific kinds of graphs visualized in FCModeler (Becker and Rojas 2001). Diva makes pluggable layout algorithms easy by separating the view logic from the layout logic.

Database and Object Properties

FCModeler allows nodes and edges in the graph model to have properties. The specific values of these properties determine the visual attributes of the corresponding node and edge figures in the view. These mappings from properties to visual attributes are encapsulated by a set of mapping rules, which can be specified in an XML file or created at run-time by the user.



node	Label	Name	Comments	organelle	category of compund
serine	serine	serine		O:cytosol	T:small molecule
cysteine	cysteine	cysteine		O:cytosol	T:small molecule
acetyl_CoA	acetyl_CoA	acetyl_CoA	C: activated acetate in cytosol	O:cytosol	T:small molecule
CoA	CoA	CoenzymeA	C:activates acetate, malate, fatty acids, etc.	O:cytosol	T:small molecule

edge	from_node...	_to_node_2	...	certainty	type	affected byc...		domain expert	location	Species
e467	dummy62	cysteine	...	1	converted			Eve Wurtele		Arabidopsis
e466	serine	dummy62	...	1	converted			Eve Wurtele		Arabidopsis
e468	dummy62	CoA	...	1	converted			Eve Wurtele		Arabidopsis
e465	acetyl_CoA	dummy62	...	1	converted			Eve Wurtele		Arabidopsis

Figure 31. The property viewer displays information about the selected nodes and edges. The properties are defined in an XML graph file generated by the relational database.

The node and link information is stored in a relational database that interacts with the graphical modeling program. The purpose of this database is to store information such as links and nodes data, search results, literature sources, and microarray data in a searchable database to support development of the Gene Expression Toolkit. This system will be used to model the structure of metabolic networks using data provided by users. It will also track the results from the tests. Figure 31 shows a property window that displays the database information about the highlighted nodes and links.

Animation

The visual attributes of the node and edge figures can be changed over time, producing an animation of the graph view. This animation consists of discrete time steps, each having a set of mapping rules. An animation controller in FCModeler applies the mapping rules to the node and edge figures for each time step in order, with a configurable delay between time steps. The node and edge figures are set back to a permanent state at the beginning of each time step, and then the new mapping rules are applied to all figures in the view. Thus, the mappings only last for a single time step, and then the figures revert back to their previous state. The user specifies the sets of mapping rules for each time step of the animation in an XML file. This file is similar to the attributes XML file, but with the addition of time step tags. Users can produce these animation files to show how the nodes interact with each other in the graph.

Metabolic Network Modeling using Fuzzy Cognitive Maps

The FCModeler tool models regulatory networks so that important relationships and hypotheses can be mined from the data. Some types of models that have been studied for representing gene regulatory networks are Boolean networks (Liang, Fuhrman et al. 1998; Akutsu, Miyano et al. 1999), linear weighting networks (Weaver, Workman et al. 1999), differential equations (Tomita, Hashimoto et al. 1999; Akutsu 2000) and Petri nets (Matsuno 2000). Circuit simulations and differential equations such as those used in the E-cell project require detailed information that is not yet known about the regulatory mechanisms between genes. Another problem is the numerical instability inherent in solving large networks of differential equations. Boolean networks analyze binary state transition matrices to look for patterns in gene expression. Each part of the network is either on or off depending on whether a signal is above or below a pre-determined threshold. These network models lack feedback. Linear weighting networks have the advantage of simplicity since they use simple weight matrices to additively combine the contributions of different regulatory elements. However, the Boolean and weighting networks are feedforward systems that cannot model the feedback present in metabolic pathways. Petri nets can handle a wide variety of information, however their complexity does not scale up well to systems that have both continuous and discrete inputs (Alla and David 1998; Reisig and Rozenberg 1998).

Fuzzy cognitive maps (FCMs) have the potential to answer many of the concerns that arise from the existing models. Fuzzy logic allows a concept or gene expression to occur to a degree—it does not have to be either on or off (Kosko 1986). FCMs have been successfully applied to systems that have uncertain and incomplete models that cannot be expressed compactly or conveniently in equations. Some examples are modeling human psychology (Hagiwara 1992), and on-line fault diagnosis at power plants (Lee, Kim et al. 1996). All of these problems have some common features. The first is the lack of quantitative information on how different variables interact. The second is that the direction of causality is at least partly known and can be articulated by a domain expert. The third is that they link concepts from different domains together using arrows of causality. These features are shared by the problem of modeling the signal transduction and gene regulatory networks.

We use a series of +/– links to model known and hypothesized signal transduction pathways. Another link type suggests a relationship between concepts with no implied causality. These links will be constructed by mining the literature using PathBinder and from Gene Expression Toolkit Database that contains the expert knowledge of biologists. Given the metabolic network, FCModeler contains advanced tools that:

- Locate and visualize cycles and strongly connected components of the graph.
- Simulate intervention in the network (e.g. what happens when a node is shut off) and search for critical paths and control points in the network.
- Capture information about how edges between graph nodes change when different regulatory factors are present.

Metabolic Network Modeling

Fuzzy cognitive maps are fuzzy digraphs that model causal flow between concepts or, in this case, genes, proteins, and transcription factors (Kosko 1986; Kosko 1986). The concepts are linked by edges that show the degree to which the concepts depend on each other. FCMs can be binary state systems called simple FCMs with causality directions that are +1, a positive causal connection, -1, a negative connection, or zero, no causal connection. The fuzzy structure allows the gene or protein levels to be expressed in the continuous range $[0,1]$. The input is the sum of the product of the fuzzy edge values. The system nonlinearly transforms the weighted input to each node using a threshold function or other nonlinear activation. FCMs are signed digraphs with feedback. Nodes stand for causal fuzzy sets where events occur to some degree. Edges stand for causal flow. The sign of an edge (+ or -) shows causal increase or decrease between nodes. The edges between nodes can also be time dependent functions that create a complex dynamical system. Neural learning laws and expert heuristics encode limit cycles and causal patterns. One learning method is differential Hebbian learning in which the edge matrix updates when a causal change occurs at the input (Dickerson and Kosko 1994).

Each causal node $C_i(t)$ is a nonlinear function that maps the output activation into a fuzzy membership degree in $[0,1]$. Simple or trivalent FCMs have causal edge weights in the set {-1,0,1} and concept values in {0,1} or {-1,1}. Simple FCMs give a quick approximation to an expert's causal knowledge. More detailed graphs can replace this link with a time-dependent and/or nonlinear function.

FCMs recall as the FCM dynamical system equilibrates. Simple FCM inference is matrix-vector multiplication followed by thresholding. State vectors $\mathbf{C}_n$ cycle through the FCM edge matrix **E**, that defines the edges e_{ki} where k is the upstream node and i is the downstream node. The system nonlinearly transforms the weighted input to each node C_i:

$$C_i\left(t_{n+1}\right)=S\left[\sum e_{ki}\left(t_n\right)C_k\left(t_n\right)\right]$$

S(*y*) is a monotonic signal function bounded function such as the sigmoid function:

$$S_j\left(y_j\right)=\frac{1}{1+e^{-c\left(y_j-T_j\right)}}$$

In this case *c*=1000 and T_j= 0.5 for all nodes. This is equivalent to a step function with a threshold at 0.5. The edges between nodes can also be time dependent functions that create a complex dynamical system.

Regulatory Links: The regulatory edges are modeled using a simple FCM model that assumes binary connecting edges: $e_{ki}=\{-1,1\}$ for the single edge case. When there are multiple excitatory or inhibitory connections, the weights are divided by the number of input connections in the absence of other information. As more information becomes known about details of the regulation, for example how RNA level affects the translation of the corresponding protein, the function of the link models will be updated. The regulatory nodes will also have self-feedback since the nodes stay on until they have been inhibited.

Conversion Links: Conversion relationships are modeled in different ways depending on the goal of the simulation study. The first case corresponds to investigating causal relationships between nodes. The node is modeled in the same manner as a regulatory link in which the presence of one node causes presence at the next node. When information about the rate of change in a reaction is available, a simple difference equation can model the gradually rising and falling levels of the nodes. When stoichiometric information is available, the links can be modeled as a set of mass-balance equations. The step size depends on the reaction rate and the stoichiometric relationship between the nodes.

Catalyzed Links: Catalyzed reactions add a dummy node that acts upon a conversion link. This allows one link to modify another link. In the current model, the catalyzed link is simulated by weighting the inputs into the dummy node in such a way that both inputs much be present for the node to be active. Another method of modeling catalyzed links is an augmented matrix that operates on the edges between the nodes. The catalyst node acts as a switch that allows a reaction to occur in the proper substrates are available. Since all of the compounds must be present in these links for a reaction to occurs the pieces must be modeled as a logical AND operation. This operation is commonly modeled as a minimum function, however, it can also be modeled as a product of all the input values (Kosko 1992).

Forcing functions: In biological systems such as cells, many of the metabolic network elements are always present. This is modeled as a node is active unless it is being inhibited:

$$C_i(t_{n+1}) = S\left[\sum e_{ki}(t_n) C_k(t_n) + 1\right]$$

Example of PathBinder-FCModeler Integration

This example shows how the pieces of the Gene Expression Toolkit can be used to create or update metabolic maps of a system using expert knowledge. The process starts with a map created by an expert or an existing metabolic pathway from a database such as KEGG or WIT (Kanehisa and Goto 2000; Overbeek, Larsen et al. 2000). The next step is to perform a PathBinder literature search for new relationships between the nodes of the existing graph. These relationships can then be assessed and added into the metabolic map. FCModeler models the effects of the changes for biologist user. An expert in the area of gibberellin metabolism constructed the map shown in Figure 32. Next a PathBinder Query is performed as shown below.

Query: Find sentences containing (either gibberellin, gibberellins, or GA) AND (either SPY, SPY-4, SPY-5, or SPY-7).

Sentence: "Here we describe detailed studies of the effects of two of these suppressors, spy-7 and gar2-1, on several different GA-responsive

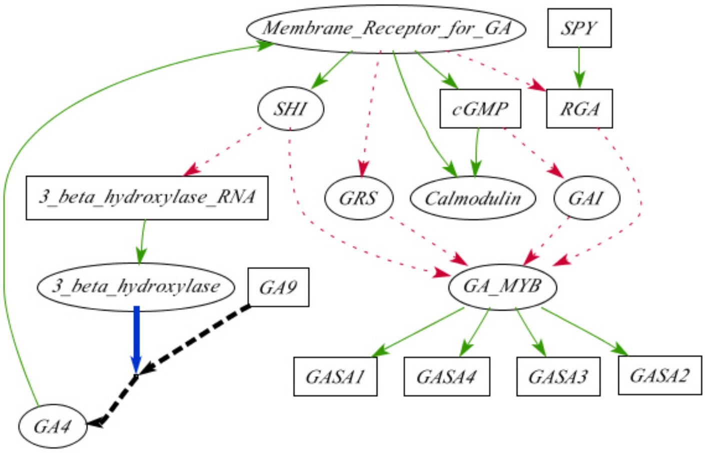


Figure 32. Hypothetical network of gibberellin metabolism and regulation in Arabidopsis. Heavy lines are catalyzed links, heavy dashed lines are conversion links, and thin lines are regulatory links. All proteins are shown in elliptical boxes.

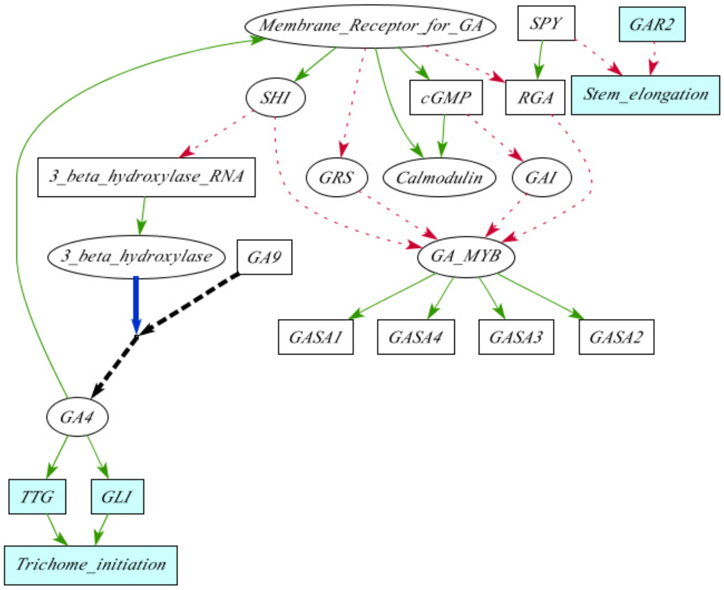


Figure 33. The updated map based on the PathBinder query result. The new nodes are shaded in.

growth processes (seed germination, vegetative growth, stem elongation, chlorophyll accumulation, and flowering) and on the in plant amounts of active and inactive GA species." Source: UI—99214450 Peng J, Richards DE, Moritz T, Cano-Delgado A, Harberd NP, Plant Physiol 1999 Apr;119(4): 1199-1208. Figure 33 shows the new graph after the information provided by the new links is added into the graph.

Example of Network Modeling

The metabolism and signal transduction of the plant hormone gibberellin in Arabidopsis (Hedden and Phillips 2000; Sun 2000) was used to test this modeling scheme. Figure 7 shows the nodes used in this test. An expert researcher in the field created the link types and causality directions. The key element in this graph is the block labeled GA4. This compound regulates many other regulatory mechanisms in plants. GAI, GRS, SPY, and GA_MYB had forcing functions applied to them. Figure 34 and Figure 35 show visualized networks at different time steps to analyze the interactions in the network. Figure 34 shows the operation of the catalyzing node, 3_beta_hydroxylase. When the node is active, GA4 is produced. These figures show how GA4 can regulate its own production through the transcription factor SHI. The result is a homeostatic control of GA4 levels. The oscillation of the GA levels directs the generation of biomolecules that, in the absence of other constraining factors, are implicated in the formation of new cellular proliferation centers, referred to as meristems. Many key features of this model, including timing, can be tested experimentally and relatively rapidly by globally monitoring temporal profiles of mRNA, protein, and metabolite.

Conclusions

The integration of a graph visualization tool with literature mining and directed searches in microarray data allows biologists to gather and combine information from the literature, their expert knowledge, and the public databases of mRNA results. Metabolic and regulatory networks can be modeled using fuzzy cognitive maps. Future plans include: simulating

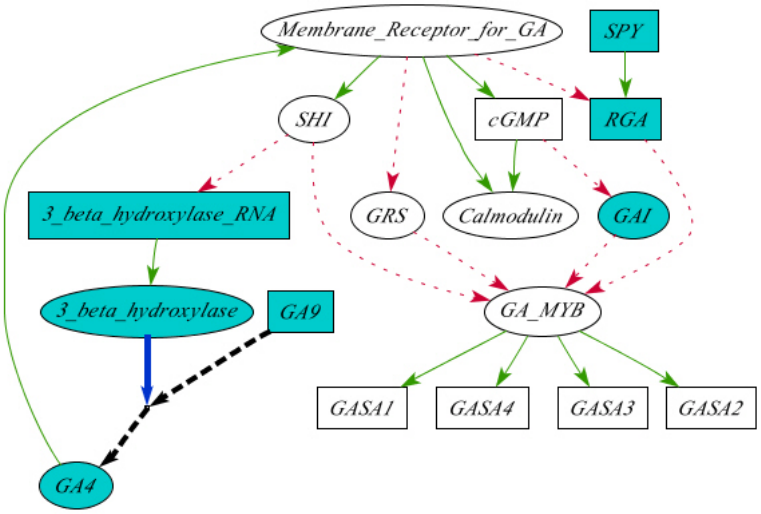


Figure 34. The catalyst, 3-beta-hydroxylase is present at this step. This allows GA9 to be converted into the active form of gibberellin, GA4. Active nodes are shaded. The nodes, SPY, GRS, and GAI are forced high in this simulation.

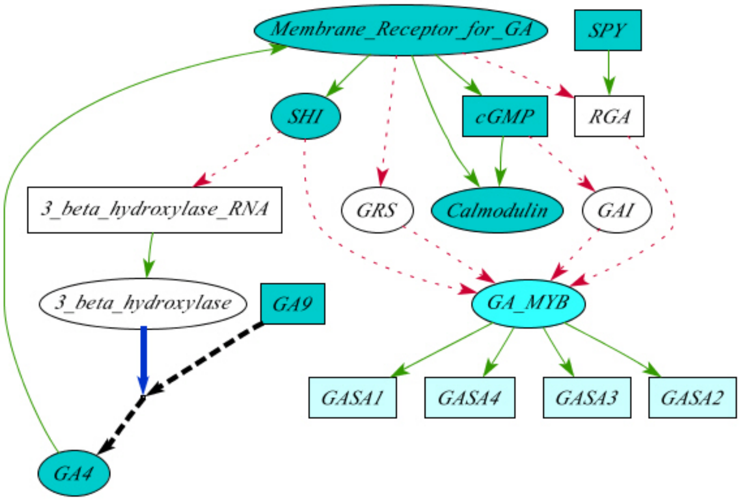


Figure 35. GA4 regulates its own production in part through the putative DNA regulatory factor SHI. SHI inhibits the 3-beta-hydroxylase-RNA, which eventually shuts down the production of GA4.

intervention in the network (e.g. what happens when a node is shut off), searching for critical paths and control points in the network, and capturing information about how edges between graph nodes change when different regulatory factors are present.

Acknowledgements

This work is supported by grants from Proctor and Gamble Corporation, the NSF (MCB-9998292), and the Plant Sciences Institute at Iowa State University.

Authors' Addresses

J.A. Dickerson, Electrical and Computer Engineering Department, Iowa State University, Ames, Iowa, USA.

D. Berleant, Electrical and Computer Engineering Department, Iowa State University, Ames, Iowa, USA.

Z. Cox, Electrical and Computer Engineering Department, Iowa State University, Ames, Iowa, USA.

W. Qi, Electrical and Computer Engineering Department, Iowa State University, Ames, Iowa, USA.

D. Ashlock, Mathematics Department, Iowa State University, Ames, Iowa, USA.

E.S. Wurtele, Botany Department, Iowa State University, Ames, Iowa, USA.

A.W. Fulmer, Proctor & Gamble Corporation, Cincinnati, Ohio, USA.

CLUSTERING CYCLES USING SELF-ORGANIZING MAPS

A paper to be submitted the Journal of Statistical Software

Zach Cox

Abstract

Cycles obtained from directed graphs may be similar to each other based on their node and edge content. Thus, clustering algorithms may be used to find natural groups of similar cycles. Prior to clustering, a distance metric and generalized set median must be defined for the objects to be clustered. Several possible representations for the cycles are discussed, including strings, graphs, and weighted sets. Weighted sets are chosen because of the simplicity of the associated metric and median. Self-organizing maps are used to find the clusters of cycles obtained from a directed graph. This approach is implemented in FCModeler, a bioinformatics tool, and uses the JSOMap package for self-organizing maps. An example with real-world data is used to show the effectiveness of the SOM in finding clusters of similar cycles.

Introduction

Many directed graphs contain cycles. Depending on the problem domain that the graph represents, these cycles may be interesting to analyze. Efficient algorithms exist for finding all of the cycles in a directed graph. Once all of the cycles are found, it may be interesting to compare the cycles based on their node and edge content. A clustering algorithm can be used to find groups of cycles that share common nodes and edges.

Traditionally, clustering algorithms are used to find groups of similar vectors. These algorithms typically use some distance metric to compare the similarity between two vectors. They also generally compute a prototype, or centroid, for each cluster using the mean of the vectors in that cluster. To use a clustering algorithm with cycles, both a distance metric and generalized set median must be defined.

Two representations for cycles immediately come to mind: strings and graphs. In the string representation, cycles are alternating sequences of nodes and edges. In the graph representation, they are graphs consisting of a node set and an edge set. Both representations have been used for pattern recognition in the past; distance metrics and generalized set medians for strings and graphs have been proposed. However, they are usually too computationally expensive to compute directly.

This paper describes the use of weighted sets as a way to represent cycles for clustering. They make the distance metric and generalized set median computations simple, while still producing good results in a clustering algorithm. A weighted set is simply a set in which the elements have non-zero real-valued weights associated with them. These sets can be computed with just like vectors of real-valued numbers; therefore we can adapt the Euclidean distance metric and statistical mean to weighted sets. This makes them easy to use in a clustering algorithm.

The self-organizing map (SOM) is used to demonstrate the use of weighted sets in finding groups of similar cycles. The SOM is an unsupervised neural network, typically used to find the cluster structure or reduce the dimensionality of a data set. The clustering is done in FCModeler using the JSOMap package, and the results of the SOM clustering are shown in an interactive graph display, where users can fully explore the results.

Cycle Search Algorithms

The cycles of a directed graph are found using a cycle enumeration, or search, algorithm. The problem of finding all of the elementary cycles in a directed graph has been well studied. Research on cycle search algorithms was done mainly in the early-middle 1970's. (Tiernan 1970; Weinblatt 1972; Tarjan 1973) published early work in this area, but used inefficient algorithms for the search problem. (Mateti and Deo 1976) provide a good overview and comparison of the various types of algorithms, with that of (Johnson 1975) being the most efficient. The algorithm is bounded by a time complexity of $O((n+e)(c+1))$ and space complexity of $O(n+e)$, where n is the number of nodes, e is the number of edges, and c is the number of elementary cycles in the graph. Thus, for each cycle, the algorithm is linear in the number of nodes and edges of the graph.

Johnson's algorithm is basically a depth-first search (DFS) that is modified to improve the search efficiency. The integer labels $\{1,2,\ldots n\}$ are first assigned to the nodes of the graph to induce an ordering, and cycles are defined to begin with the minimum labeled node. Each node s in the graph is chosen successively and the graph is searched for cycles rooted at that node. When all cycles rooted at node s are found, the next node $s+1$ is chosen and the graph is searched again.

Strongly connected components (SCC) are used to find areas of the graph that contain all cycles rooted at a particular node s. Specifically, a graph F is induced from the original graph G by the nodes $\{s,s+1,\ldots n\}$. Then, the strongly connected components of F are found, and the component K that contains node s is searched for all cycles rooted at s. SCC's have the property that all nodes are reachable from all other nodes. Thus, the SCC containing node s contains all cycles rooted at s. (Tarjan 1972) first presented an algorithm for finding all the SCC's of a directed graph in time $O(n+e)$, and (Bang-Jensen and Gutin 2001) provide a good description of the algorithm.

During the search for cycles rooted at node s, information is stored about nodes and edges that don't lead to cycles. This prevents those nodes and edges from being searched again in the future. This information is stored using blocked nodes and B-sets. Basically, nodes are blocked when they shouldn't be added to the search path and are only unblocked if the algorithm determines that they are on a cycle rooted at s. Each node also has a B-set that contains its predecessors that should be unblocked if it becomes unblocked. Blocked nodes and B-sets are key both to the algorithm's correctness (finding all elementary cycles only once) and its efficiency. For more information on the algorithm and its proof, please see (Johnson 1975).

Cycle Data Structures

For any kind of analysis to be done, the cycles must be represented somehow. There are basically three different representations for cycles: strings, graphs, and weighted sets. As seen in the following sections, each representation will have different distance metrics and generalized set medians associated with them. In all cases, C is the cycle, v is a node (or vertex), e is an edge, and k is the length of the cycle.

Strings

Since a cycle is just a path through a graph that ends at its beginning, it can be represented by a string. The string is an alternating sequence of nodes v_i and edges e_i that compose the cyclic path through the graph. The starting and ending nodes are the same, and each edge connects its surrounding nodes:

$$C = v_1 e_1 v_2 \ldots v_{k-1} e_{k-1} v_k$$
$$v_k = v_1$$
$$e_i = (v_i, v_j)$$

Note that since the cycle does not have a specific beginning or end, all cyclic permutations of the same string are equivalent to each other:

$$C = v_1 e_1 v_2 e_2 v_3 \ldots v_{k-1} e_{k-1} v_k = v_2 e_2 v_3 \ldots v_{k-1} e_{k-1} v_k e_1 v_2$$

The string can also be compressed by removing all of the nodes or edges. In this case, additional processing is required to determine the missing nodes or edges:

$$C = e_1 e_2 \ldots e_{k-1}$$
$$C = v_1 v_2 \ldots v_k$$

Graphs

Since a cycle is made up of nodes and edges that connect those nodes, it is a graph. Technically it is a subgraph of the original graph, but is a graph in it's own right. Therefore, the cycle can be represented identically to a general graph:

$$C = (V = \{v_1, v_2, \ldots, v_{k-1}\}, E = \{e_1, e_2, \ldots, e_{k-1}\})$$

Weighted Sets

A vector space can be defined by the original graph, with one dimension for each node and edge of the graph. Any cycle (or subgraph for that matter) is then a vector of ones and zeros, with a one representing the existence of that node or edge in the cycle, and a zero the absence of that node or edge. Real numbers can also be used in place of ones and zeros, where a zero represents the absence of the node or edge and a non-zero real number represents the weighted presence of the node or edge. In this representation, cycles are vectors, and any computations like euclidean distance or statistical average can be done with them.

In general, these cycle vectors will contain many zero terms: that is, most cycles consist of a small subset of the nodes and edges of the graph. Weighted sets enhance the efficiency of the vector space representation. A weighted set is just a normal set in which each element *e* has an associated non-zero real-valued weight *w*:

$$S = \{(e_1, w_1), (e_2, w_2), \ldots, (e_n, w_n)\}$$
$$e_i \in \Sigma$$
$$w_i \in \Re$$

The operator [], when applied to a weighted set, returns the weight of a specified element.

$$S[e_i] = \begin{cases} w_i & \text{if } e_i \in S \\ 0 & \text{otherwise} \end{cases}$$

Similarly, a new weight can be assigned to a specified element by using the [] operator.

$$S[e_i] = w_{new}$$

Weighted sets are basically just vectors without the zeros; computations can be done with them in the same way as regular vectors:

$$S_1 + S_2 = \{(e_1, S_1[e_1] + S_2[e_1]), \ldots, (e_n, S_1[e_n] + S_2[e_n])\}$$

A weighted graph *WG* can be created from a regular graph *G* by using weighted sets in place of the node and edge sets, and initializing all of the weights to one:

$$G = (\{v_1, v_2, \ldots, v_{k-1}\}, \{e_1, e_2, \ldots, e_{k-1}\}) \rightarrow WG = (\{(v_1,1), (v_2,1), \ldots, (v_{k-1},1)\}, \{(e_1,1), (e_2,1), \ldots, (e_{k-1},1)\})$$

Operations on weighted graphs are similar to those on weighted sets:

$$G_1 + G_2 = (V_1 + V_2, E_1 + E_2)$$

Distance Metrics

A metric compares two objects and calculates the distance or dissimilarity between them. A metric function *d* must satisfy the following conditions:

1. $d(a,b) = 0 \Leftrightarrow a = b$
2. $d(a,b) = d(b,a)$
3. $d(a,b) + d(b,c) \leq d(a,c)$

Distance metrics are fundamental to clustering algorithms and can be defined for all three cycle representations.

Strings

The so-called Levenshtein edit distance (Wagner and Fischer 1974) is the most well-known distance metric for strings. It is commonly used in spell-checking applications, where it compares strings of alphabetic characters. The distance between two strings is defined as the minimum number of insertions, deletions, and substitutions of characters that are required to transform one string into the other string. For example, the distance between the words ‘true’ and ‘truth’ is two: substitute ‘t’ for ‘e’ and insert ‘h’ at the end of the first string to transform it into the second string. (Wagner and Fischer 1974) give an efficient dynamic programming-based algorithm that runs in $O(nm)$ time, where n and m are the lengths of the two strings.

This distance metric faces several problems when applied to cycles. First, not all possible strings form valid cycles in the given graph. In other words, the graph structure determines which nodes or edges can be inserted, deleted, or substituted. This adds additional complexity to the algorithm and its running time.

As stated previously, these are cyclic strings. That is, all cyclic permutations of the same string are equivalent to each other. This presents a problem when computing the edit distance between two strings. The edit distance must now be restated as the minimum number of insertions, deletions, and substitutions needed to transform one string into any cyclic permutation of the other string. (Maes 1990; Bunke and Buhler 1993; Gregor and Thomason 1993; Marzal and Barrachina 2000; Mollineda, Vidal et al. 2000) present various exact and approximate metrics for comparing cyclic strings. All of these algorithms are much more complex and worse in time complexity than the Levenshtein metric for linear strings.

Graphs

Determining the similarity of graphs, or graph matching, is also a well-studied area in pattern recognition, where graphs are used to abstractly represent different objects. In the literature, there are two main approaches to computing the distance between two graphs: graph edit distance and maximal common subgraph (Bunke 2000).

Graph edit distance is basically the string edit distance introduced earlier applied to graphs. The edit distance between two graphs is defined as the minimum number of node and edge insertions, deletions, and substitutions that are required to transform one graph into the other (Bunke 1999; Jiang, Munger et al. 2001). The algorithms for computing graph edit distance are known to be NP-complete.

The maximal common subgraph of two graphs offers another way to compute the distance between two graphs (Bunke and Shearer 1998). A graph G is a common subgraph of two other graphs G_1 and G_2 if G is isomorphic to a subgraph of G_1 and a subgraph of G_2. G is maximal if there is no other common subgraph G' that contains more nodes than G. Algorithms have been presented that run in $O\left((nm)^n\right)$ and $O\left(2^n\right)$ time.

Note that in (Bunke and Shearer 1998), nodes and edges are allowed to have the same labels, which results in the poor computational efficiency. If a restriction is placed on the graph such that all nodes and edges have distinct labels, the computational complexity can be reduced. The distance metric is then given by:

$$d(G_1,G_2)=1-\frac{|\mathrm{mcs}(G_1,G_2)|}{\max(|G_1|,|G_2|)}=1-\frac{|G_1\cap G_2|}{\max(|G_1|,|G_2|)}$$
$$G_1\cap G_2=(V_1\cap V_2,E_1\cap E_2)$$
$$|G|=|V|+|E|$$

Weighted Sets

As stated earlier, weighted sets are equivalent to vectors of real numbers, so pair wise operations such as addition, subtraction, multiplication, and division are easily defined. The distance between two weighted sets can then be defined similarly to the euclidean distance between two vectors:

$$d(S_1,S_2)=\sqrt{\sum_{e\in(S_1\cup S_2)}(S_1[e]-S_2[e])^2}$$

Likewise, the distance between two weighted graphs is simply the sum of the distance between the two node sets and the distance between the two edges sets:

$$d(G_1,G_2)=d(V_1,V_2)+d(E_1,E_2)$$

Generalized Set Median

The median of a set of objects is defined as the element of the set that minimizes the sum of distances between that element and all other elements in the set. The generalized median of a set of objects is some element of the set of all possible objects that minimizes the sum of distances between that element and all other elements in the set. Clustering algorithms generally compute a prototype for each cluster of objects. In the case of vectors, this prototype is simply the arithmetic mean of all the vectors in the cluster. To use a clustering algorithm with cycles, we therefore need a suitable generalized median for a set of cycles, depending on the specific cycle representation.

Strings

Let Σ be some alphabet of characters, and let Σ^* be the set of all finite-length strings of characters from Σ. Then, the generalized median $\bar{s}$ of a set of strings $S = \{s_1, s_2, \ldots, s_n\}$ is defined as:

$$\bar{s} = \arg\min_{s \in \Sigma^*} \sum_{i=1}^{n} \mathrm{d}(s, s_i)$$

That is, $\bar{s}$ is the string in the set of all finite-length strings that minimizes the sum of distances between it and the strings in the set *S*. Generally, finding the string $\bar{s}$ requires computing the distance between every string in Σ^* and the set *S*, resulting in an NP-complete problem (Higuera and Casacuberta 2000). Faster algorithms have been proposed (Kohonen 1985; Martinez-Hinarejos, Juan et al. 2000) that produce approximate solutions to the generalized median string.

Graphs

Given some set of possible nodes and a set of possible edges let *U* be the set of all possible graphs, such that each node and edge can be used only once. Then, the generalized median $\bar{g}$ of a set of graphs $G = \{g_1, g_2, \ldots, g_n\}$ is defined as:

$$\bar{g} = \arg\min_{g \in U} \sum_{i=1}^{n} \mathrm{d}(g, g_i)$$

That is, $\bar{g}$ is the graph in the set of all possible graphs that minimizes the sum of distances between it and the graphs in the set *G*. Like the generalized median string problem, finding the generalized median graph is NP-complete (Bunke 2000). Methods for finding approximate solutions have been proposed using other techniques such as genetic algorithms (Jiang, Munger et al. 2001).

Weighted Sets

The generalized median $\bar{s}$ of a set of weighted sets $S = \{s_1, s_2, \ldots s_n\}$ can be adapted from the definition of the statistical mean of a set of vectors:

$$\bar{s} = \frac{\sum_{i=1}^{n} s_i}{n}$$

Likewise, the generalized median $\bar{g}$ of a set of weighted graphs $G = \{g_1, g_2, \ldots, g_n\}$ is given as:

$$\bar{g} = \frac{\sum_{i=1}^{n} g_i}{n}$$

Obviously, computing the generalized median of a set of weighted graphs is much simpler than of a set of strings or graphs as shown earlier. Basically it amounts to creating the union of all the graphs in *G*, summing the weights of the nodes and edges and dividing by n.

Using Self-Organizing Maps to Find Clusters of Cycles

The SOM (Kohonen 2001) is an unsupervised neural network and can be used for non-parametric regression, non-linear projection, dimensionality reduction, feature extraction, clustering, and visualization depending on the interpretation and use of its results. Figure 15 below shows a typical architecture of the neural network.

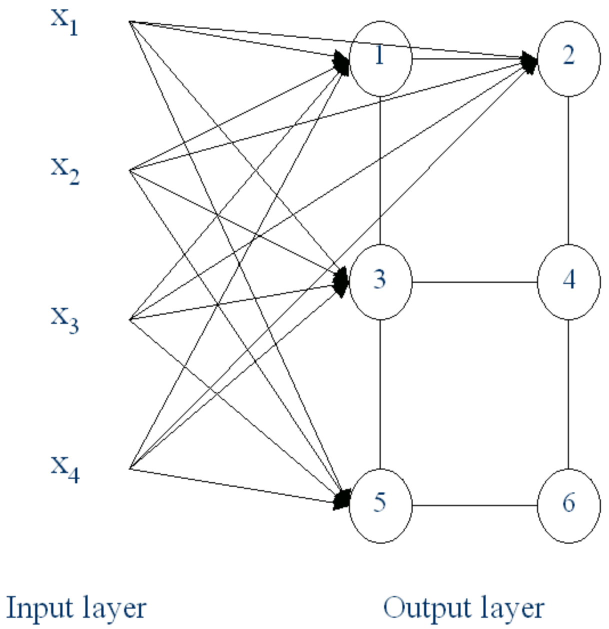


Figure 36. The SOM as an unsupervised neural network. The SOM consists of an input layer, where the input patterns are presented, and an output layer of connected neurons. These neurons exist as discrete points in some space, in this case 2-D, and each neuron has a weight vector of the same dimensionality as the input patterns.

Each neuron, or *node*, in the output layer has two vectors associated with it that exist in two different spaces: feature space and input space. Feature space is the space in which the nodes exist (which is typically 1-D or 2-D) and input space is the space in which the data exists (typically 2-D or higher). The first vector describes the position of the node in feature space, and the second represents the weight vector of the neuron. Each neuron has a weight associated with each input variable; the lines in Figure 15 represent these weights, and together they form a vector of the same dimensionality as the input data. These weight vectors can also be represented as points in input space; in this case, they are referred to as *models* since they are the input space representation of the nodes. When referring to the map as a neural network, it is natural to refer to neurons and weight vectors; when referring to points in feature space and input space, nodes and models are used instead.

The goal of the SOM algorithm is to adjust the weight vectors of the neurons to match the training data. One can also think of this as positioning the models of the nodes in input space to match the distribution of the data. The algorithm attempts to do this so that nodes near each other in feature space will have models near each other in input space. Positioning the models to approximate the data distribution can be thought of as *non-parametric regression*, much like discretized principal curves (Hastie and Stuetzle 1989; Mulier and Cherkassky 1995). The trained map can then perform *non-linear projection* of data points in input space, by finding the nearest model to the data point and projecting the point onto the model's node in feature space. For a more detailed description of the SOM algorithm, please see (Kohonen 2001).

There are many different algorithms that can be used to position the models of the map in the input space of the data. One commonly used algorithm, called the Batch SOM algorithm, is as follows:

1. Assign each pattern to the node with the most similar model
$$V_j = \left\{ \mathbf{x}_p \middle| \mathrm{d}(\mathbf{x}_p, \mathbf{m}_j) = \min_i \mathrm{d}(\mathbf{x}_p, \mathbf{m}_i) \right\}$$
2. Update the models of the map using the sets V_j, medians $\bar{\mathbf{x}}_j$, and neighborhood function $\mathrm{K}(\ )$.
$$\mathbf{m}_i = \frac{\sum_j \bar{\mathbf{x}}_j |V_j| \mathrm{K}(\mathbf{n}_i, \mathbf{n}_j)}{\sum_j |V_j| \mathrm{K}(\mathbf{n}_i, \mathbf{n}_j)}$$
3. Decrease the width of the neighborhood function and return to step 1

As can be seen in the algorithm above, both a distance metric function $\mathrm{d}(\ )$ and a generalized median $\bar{\mathbf{x}}$ are required. Since the patterns being compared in this case are cycles represented by weighted graphs, the distance metric and generalized median presented earlier can be used. With these two functions, the SOM can now be used to find the clusters of cycles obtained from a directed graph.

Visualizing the Results

A good deal of research has been done on SOM visualization (Himberg 1998; Kaski, Venna et al. 1999; Vesanto 1999; Cox to be submitted). However, this previous work has concentrated on the use of the SOM with traditional numerical vectors. Since this paper focuses on clustering cycles and not numerical vectors, a custom approach was needed for visualizing the results.

The cycle clustering is implemented as a part of FCModeler, a software tool used to visualize, analyze, and simulate metabolic networks (Dickerson, Berleant et al. 2001; Dickerson, Cox et al. 2001; Dickerson, Berleant et al. to be published in 2002). Figure 37 below shows a large metabolic network graph in FCModeler. First, Johnson's algorithm (Johnson 1975) is used to find all of the cycles in the directed graph, then the cycles are organized into clusters by the SOM. The JSOMap package (Cox 2002) provided support for using the SOM.

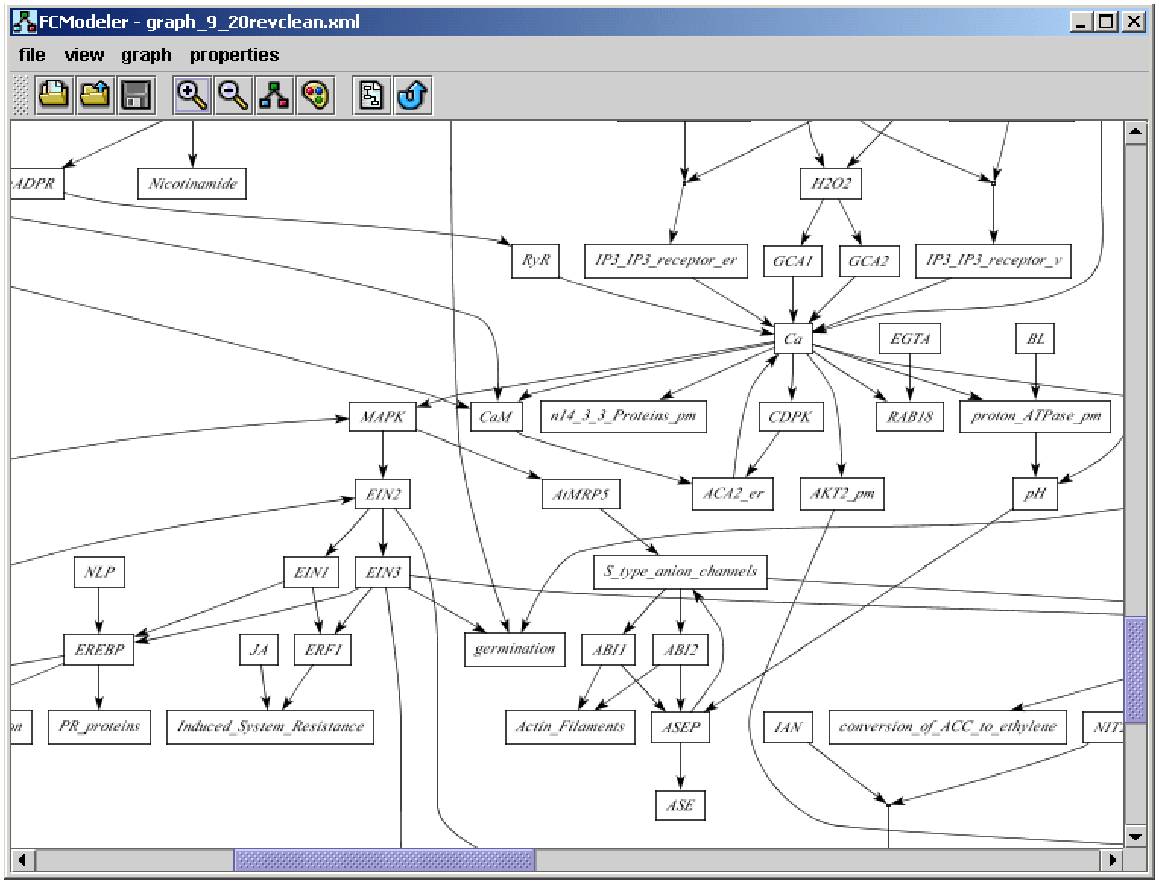


Figure 37. Example of a large metabolic network graph shown in FCModeler.

Once the SOM algorithm is finished, the results are viewed in another window. This map view window shows a grid of small graphs, one for each unit of the map. Each graph represents the model for that particular map unit. The models are just the generalized median of the set of cycles assigned to that map unit, as presented earlier. Figure 38 below shows an example of a 4-by-4 SOM applied to the graph in Figure 37. Node color and edge thickness are used to show the weights of the nodes and edges of the models. Brighter red corresponds to higher weights of the nodes, while thicker lines correspond to higher weights of the edges.

As Figure 38 shows, it can be a bit difficult to fully examine the model graphs in the map view. Clicking on one of the graphs in the map view puts it in an additional FCModeler window for easier analysis. A good cluster of cycles can be seen in Figure 39 below, which is a view of the model graph in the bottom-left map unit of Figure 38. This model represents ten individual cycles. All cycles contain the nodes ACC_synthase, ACC, ethylene, CTR1, EIN2, EIN3, senescence, NIT2, and IAA. Note the dark red color of these nodes and the edges between them, showing the maximum weight value. Also, note the five different paths between ethylene and CTR1, and the two different paths between CTR1 and EIN2. There are two cycles for each of the five different paths between ethylene and CTR1: one path going from CTR1 directly to EIN2 and another going through MAPKK and MAPK. This is where the ten cycles come from. Obviously, the SOM correctly grouped together these ten cycles into one cluster.

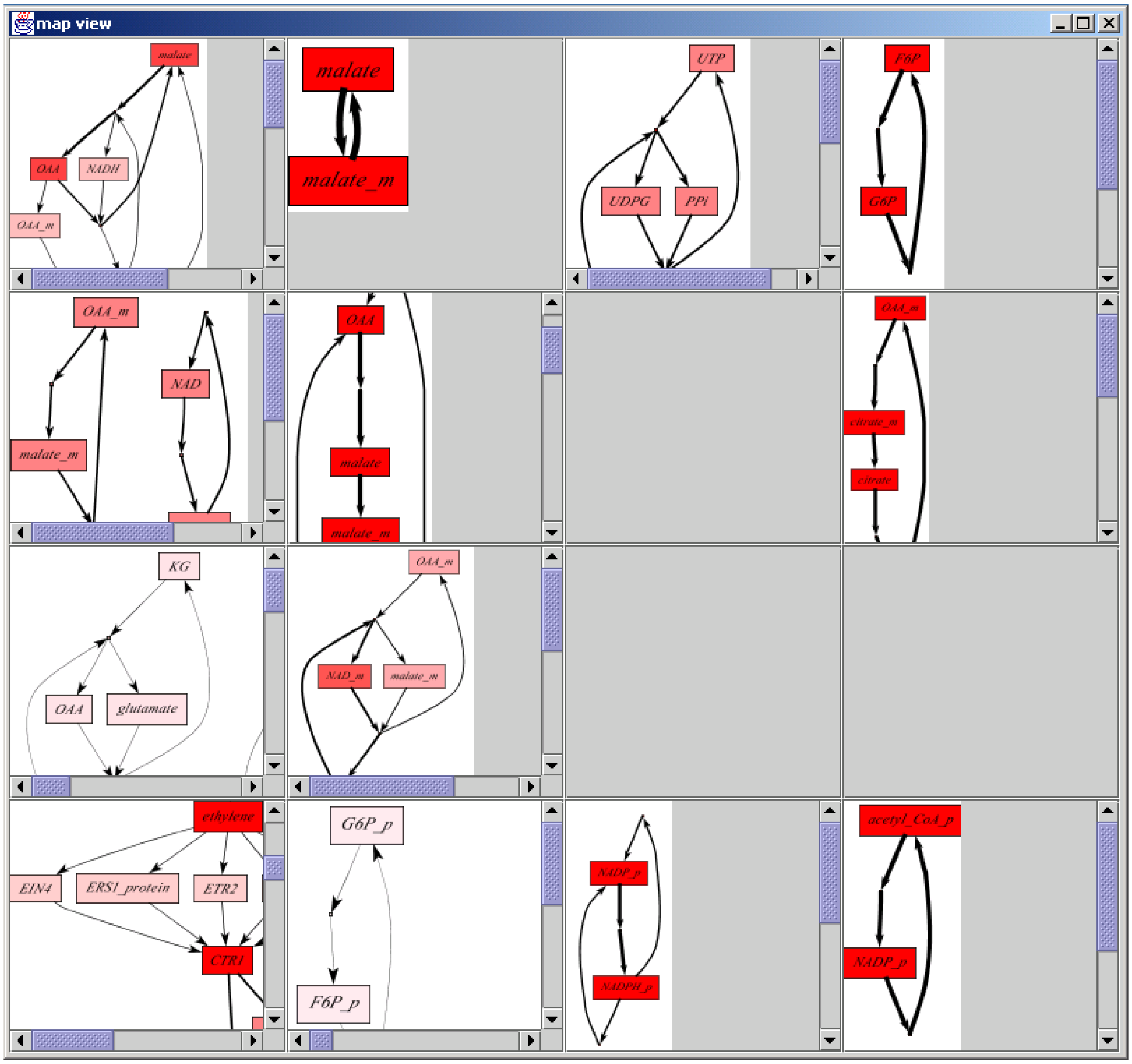


Figure 38. Map view showing results of the SOM algorithm. Each graph shows the model of the corresponding map unit, which is the generalized median of the cycles assigned to that map unit.

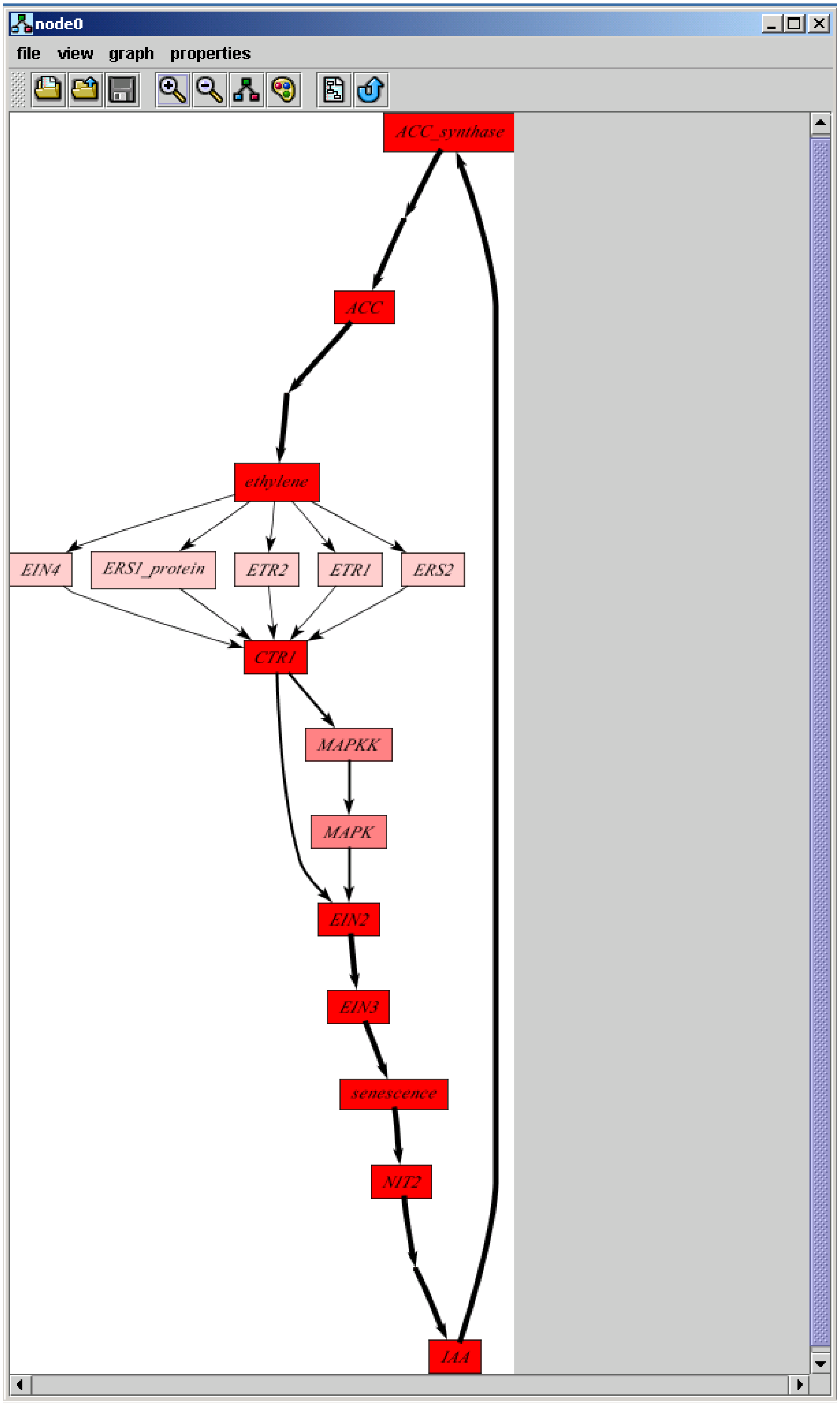
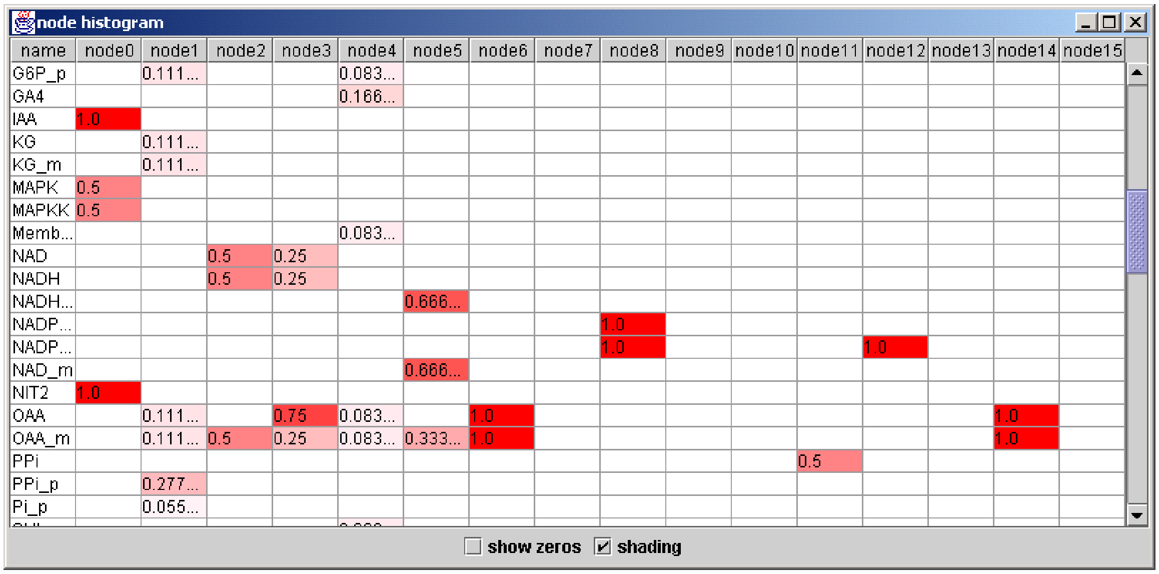


Figure 39. Full view of the bottom-left map unit, showing ten cycles clustered together.

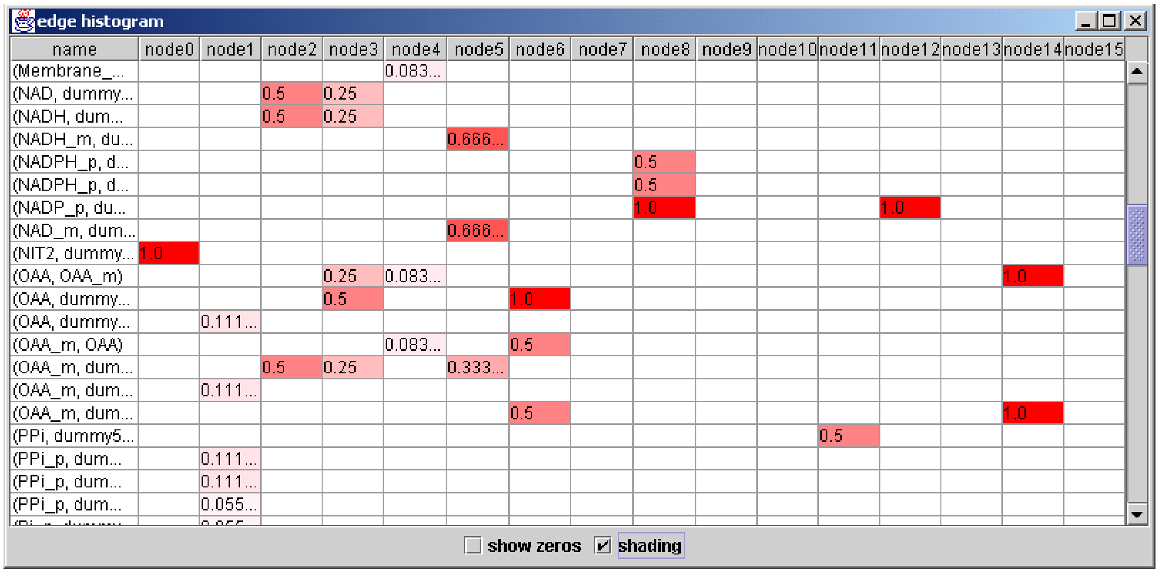
In addition to the map view and individual model view, tables showing all of the node and edge weights are also created. Each column in the table represents one map unit, while each row represents one node or one edge. The entries in the table are the weights of each node or edge in each map unit. The node and edge tables for the example discussed in this section are shown below in Figure 40 and Figure 41. To make visualization easier, the zero entries can be removed from the table, and the cells can be rendered based on the weight (with exactly the same color scheme as the node coloring).



name	node0	node1	node2	node3	node4	node5	node6	node7	node8	node9	node10	node11	node12	node13	node14	node15
G6P_p		0.111...			0.083...											
GA4					0.166...											
IAA	1.0															
KG		0.111...														
KG_m		0.111...														
MAPK	0.5															
MAPKK	0.5															
Memb...					0.083...											
NAD			0.5	0.25												
NADH			0.5	0.25												
NADH...						0.666...										
NADP...									1.0							
NADP...									1.0				1.0			
NAD_m						0.666...										
NIT2	1.0															
OAA		0.111...		0.75	0.083...		1.0								1.0	
OAA_m		0.111...	0.5	0.25	0.083...	0.333...	1.0								1.0	
PPi												0.5				
PPi_p		0.277...														
Pi_p		0.055...														

☐ show zeros ☑ shading

Figure 40. Node weights for the example presented in this section.



name	node0	node1	node2	node3	node4	node5	node6	node7	node8	node9	node10	node11	node12	node13	node14	node15
(Membrane_...					0.083...											
(NAD, dummy...			0.5	0.25												
(NADH, dum...			0.5	0.25												
(NADH_m, du...						0.666...										
(NADPH_p, d...									0.5							
(NADPH_p, d...									0.5							
(NADP_p, du...									1.0				1.0			
(NAD_m, dum...						0.666...										
(NIT2, dummy...	1.0															
(OAA, OAA_m)				0.25	0.083...										1.0	
(OAA, dummy...				0.5			1.0									
(OAA, dummy...		0.111...														
(OAA_m, OAA)					0.083...		0.5									
(OAA_m, dum...			0.5	0.25		0.333...										
(OAA_m, dum...		0.111...														
(OAA_m, dum...							0.5								1.0	
(PPi, dummy5...												0.5				
(PPi_p, dum...		0.111...														
(PPi_p, dum...		0.111...														
(PPi_p, dum...		0.055...														

☐ show zeros ☑ shading

Figure 41. Edge weights for the example presented in this section.

Conclusion

This paper demonstrated the clustering of cycles using self-organizing maps and effective ways of visualizing the results. Johnson's cycle enumeration algorithm was briefly discussed and serves as the search algorithm for finding all of the cycles in a directed graph. Several possible representations for the cycles were discussed, including strings, graphs, and weighted sets. Based on the available distance metrics and generalized medians, weighted sets were chosen because of the simplicity of computing with them. An example with real-world data showed the effectiveness of the SOM in finding clusters of similar cycles, as seen in the visualizations created by the software.

GENERAL CONCLUSIONS

General Discussion

Unsupervised Learning

Unsupervised learning algorithms are often ad-hoc, exploratory methods for analyzing a high dimensional data set. It is not always obvious if the results of unsupervised learning are good or even meaningful. Exploratory data analysis techniques like dynamic graphics offer new ways of visualizing unsupervised learning algorithms. These methods allow the user to view how the algorithms work, visualize their results in the high dimensional space of the data, and interact with the plots to customize the information shown.

For all three of the unsupervised learning algorithms described in this thesis, the grand tour shows the progression of the algorithm and its results in the high dimensional space of the data. Viewing the process of the algorithms with the data offers an understanding of how they work. By visualizing their results in the data space, confidence in the algorithms can be increased (or decreased).

The amount of information calculated by the fuzzy c-means clustering algorithm is too great to show effectively on a single static plot of the data all at once. Four different methods for reducing this information to a displayable amount were presented. These methods calculate shapes and colors based on the fuzzy membership matrix and render them onto the grand tour view in real-time, to provide a better understanding of the results of the fuzzy c-means algorithm. The user can also interact with the view to change and customize how the membership information is shown.

In addition to the grand tour view, a map view was used for the self-organizing map visualization. This map view shows the 2-dimensional grid of points that the SOM algorithm stretches around the high dimensional input space of the data. The appearance of the two views was linked so that changes in one view affected the other as well. This linked brushing aids the user in exploring the mapping from input space to feature space calculated by the SOM algorithm by seeing how data points in input space are projected onto the map in feature space.

Graph Visualization

FCModeler not only provides a good environment for exploring metabolic networks, it is a general graph visualization and analysis tool. By utilizing interactive graphics, users can freely modify both the underlying graph structure and the visualization itself. New layout algorithms can be seamlessly integrated into the software to give multiple perspectives of a graph. The dynamic property to visual attribute mapping functionality of FCModeler allows the user to visualize the multiple categorical variables of the nodes and edges in an intuitive way.

Cycle Clustering

Cycle clustering is an interesting application of unsupervised learning, dynamic graphics, and graph visualization. Of the three different representations reviewed, weighted sets were chosen as the data structure for the clustering based on the simplicity of their associated distance metric and generalized set median. Visualization of the results proved challenging, as the basic patterns being clustered (the cycles) are non-traditional data objects. Custom visualizations, including the map view showing the clusters of cycles and the node and edge weight tables, presented the results of the self-organizing maps in an understandable manner.

Recommendations for Future Research

Unsupervised Learning

Data visualization is just starting to find uses in data mining (Fayyad, Grinstein et al. 2002). Thus, there are plenty of areas that need additional work and many new ideas to try out. Instead of covering everything, only ideas for future research related to unsupervised learning algorithms will be discussed in this section.

The grand tour as used in this research creates random 2-D projections of the data and smoothly rotates between them. However, projection pursuit can be used with the grand tour to search for more interesting projections (Cook, Buja et al. 1995). Perhaps new projection pursuit indices could be created based on the results of the unsupervised learning algorithms, providing better tours over the data.

The grand tour is definitely not the only method for visualizing high dimensional data. Many other multivariate visualization techniques exist, including scatterplot matrices, parallel coordinates, radviz, etc. (Hoffman and Grinstein 2002) provide a good overview of the many available methods. One or several other multivariate views could be used together with the grand tour to show the data from another angle. Using linked brushing, all of the views could show the same information and support interaction.

Several other SOM-based visualizations were discussed in the literature review. Although they are static and focus mainly on the map view, it would be interesting to combine them with the high dimensional, interactive techniques presented in this thesis. Also, the SOM is not restricted to a 2-dimensional feature space even though that is primarily what is used in practice. A multivariate view could be used to show the SOM in the high dimensional feature space, much like the 2-D map view.

These visualization methods are not restricted to the three unsupervised learning algorithms used in this research. Using a high dimensional view to see the model along with the data and interactive plots are general ideas that could be used with other learning algorithms, such as model-based clustering, hierarchical clustering, self-supervised multi-layer perceptron neural networks, etc. Also, these techniques may prove useful for visualizing supervised learning algorithms as well, such as neural networks, support vector machines, CART, etc. In this case, the visualizations may focus more on showing the decision function created by the learning algorithm.

Cycle Clustering

Although weighted sets turned out to be a good data representation for the cycles, it would be interesting to implement both the string and graph representations as well. Of course, this would also require the implementation of the associated metrics and medians, which in most cases are known to be NP-complete problems. The approximate solutions, discussed in the previous chapter, would be needed to make the approach tractable. Also, one could define a measure of similarity not on specific node and edge content of the cycles, but on the values of the multiple variables of the nodes and edges. This would allow the search for cycles with similar properties, not with the same nodes and edges.

APPENDIX: ACCOMPANYING CD-ROM AND OPERATING INSTRUCTIONS

System requirements: Any operating system capable of running Java software; approximately 150MB hard disk space; 64MB minimum RAM

CD-ROM contains the following software packages:

- Orca – Java-based data visualization package
- JSOMap – Java-based self-organizing maps package
- FCModeler – Java-based graph visualization and analysis package

Since the software is written in Java, java virtual machines and installation instructions are provided for the following operating systems, located in the jdk directory:

- Windows 95, 98 (1st or 2nd edition), NT 4.0 with Service Pack 5, ME, 2000 Professional, 2000 Server, 2000 Advanced Server, or XP
- Linux
- Solaris

For information on running the applications, please see the file Instructions.txt included on the CD-ROM.

REFERENCES

Akutsu, T., S. Miyano, et al. (1999). Identification of Genetic Networks from a Small Number of Gene Expression Patterns Under the Boolean Network Model. Pacific Symposium on Biocomputing 4, Hawaii.

Akutsu, T., Miyano, S., Kuhara, S. (2000). Algorithms for Inferring Qualitative Models of Biological Networks. Pacific Symposium on Biocomputing 5, Hawaii.

Alla, H. and R. David (1998). "Continuous and Hybrid Petri Nets." Journal of Circuits, Systems, and Computers **8**(1): 159-188.

Andrade, M. A. and A. Valencia (1998). "Automatic extraction of keywords from scientific text: Application to the knowledge domain of protein families." Bioinformatics **14**: 600-607.

Angel, E. (2000). Interactive Computer Graphics. Massachusetts, Addison Wesley Longman.

Angeline, P. J. (1996). Advances in Genetic Programming II. Cambridge, Mass., MIT Press.

Asimov, D. (1985). "The Grand Tour: A Tool for Viewing Multidimensional Data." SIAM Journal of Scientific and Statistical Computing **6**(1): 128-143.

Bang-Jensen, J. and G. Gutin (2001). Digraphs: Theory, Algorithms, and Applications. New York, Springer.

Becker, M. and I. Rojas (2001). "A graph layout algorithm for drawing metabolic pathways." Bioinformatics **17**: 461-467.

Becker, R. A. and W. S. Cleveland (1987). Brushing Scatterplots. Dynamic Graphics for Statistics. M. E. McGill. Monterey, CA, Wadworth**:** 201-224.

Becker, R. A., W. S. Cleveland, et al. (1988). Dynamic Graphics for Data Analysis. Dynamic Graphics for Statistics. M. E. McGill. Monterey, CA, Wadsworth**:** 1-50.

Berleant, D. (1995). "Engineering Word Experts for Word Disambiguation." Natural Language Engineering **1**(4): 339-362.

Bezdek, J. C. (1981). Pattern Recognition with Fuzzy Objective Function Algorithms. New York, Plenum Press.

Bloch, J. (2001). Effective Java: Programming Language Guide. Boston, Addison-Wesley.

Brown, M. P. S., W. N. Grundy, et al. (2000). "Knowledge-based analysis of microarray gene expression data by using support vector machines." Proceedings National Academy of Science **97**(1): 262-267.

Buja, A., D. Cook, et al. (1997). "Dynamic Projections in High-Dimensional Visualization: Theory and Computational Methods." Technical Report. AT&T Labs. Florham Park, NJ.

Buja, A., D. Cook, et al. (1996). "Interactive High-Dimensional Data Visualization." Journal of Computational and Graphical Statistics **5**(1): 78-99.

Bunke, H. (1999). "Error Correcting Graph Matching: On the Influence of the Underlying Cost Function." IEEE Transactions on Pattern Analysis and Machine Intelligence **21**(9): 917-922.

Bunke, H. (2000). Recent Developments in Graph Matching. Proceedings of the 15th International Conference on Pattern Recognition, Barcelona, Spain.

Bunke, H. and U. Buhler (1993). "Applications of Approximate String Matching to 2D Shape Recognition." Pattern Recognition **26**(12): 1797-1812.

Bunke, H. and K. Shearer (1998). "A graph distance metric based on the maximal common subgraph." Pattern Recognition Letters **19**: 255-259.

Cherkassky, V. and F. Mulier (1998). Learning From Data. New York, John Wiley & Sons, Inc.

Cleveland, W. S. and M. E. McGill (1988). Dynamic Graphics for Statistics. Monterey, CA, Wadsworth.

Collier, N. H., S. Park, N. Ogata, U.Y. Tateishi, C. Nobata, T. Ohta, T. Sekimizu, H. Imai, K. Ibushi, and J. Tsujuii (1999). The GENIA project: corpus-based knowledge acquisition and information extraction from genome research papers. European Association for Computational Linguistics (EACL) Conference.

Cook, D. and A. Buja (1997). "Manual Controls For High-Dimensional Data Projections." Journal of Computational and Graphical Statistics **6**(4): 464-480.

Cook, D., A. Buja, et al. (1995). "Grand Tour and Projection Pursuit." Journal of Computational and Graphical Statistics **4**(3): 155-172.

Cox, Z. (February 28, 2002). JSOMap: a Java-based Self-Organizing Map package. http://jsomap.sourceforge.net. Date accessed: June 18, 2002.

Cox, Z. (to be submitted). "Visual Exploration of Self-Organizing Maps Using the Grand Tour and Linked Brushing." Journal of Statistical Software.

Dickerson, J. A., D. Berleant, et al. (to be published in 2002). Creating and Modeling Metabolic and Regulatory Networks Using Text Mining and Fuzzy Expert Systems. Computational Biology and Genome Informatics. J. T. L. Wang, World Scientific.

Dickerson, J. A., D. Berleant, et al. (2001). Creating Metabolic Network Models using Text Mining and Expert Knowledge. Atlantic Symposium on Molecular Biology and Genome Information Systems and Technology (CBGIST 2001), Durham, North Carolina.

Dickerson, J. A., Z. Cox, et al. (2001). Creating Metabolic and Regulatory Network Models using Fuzzy Cognitive Maps. North American Fuzzy Information Processing Conference (NAFIPS), Vancouver, B.C.

Dickerson, J. A. and B. Kosko (1994). "Virtual Worlds as Fuzzy Cognitive Maps." Presence **3**(2, Spring): 173-189.

Dickerson, J. A., R. Matalon, et al. (1998). Fuzzy Clustering of Vitamin and Homocysteine Levels in Patients with a History of Ischemic Stroke. Conference of the North American Fuzzy Information Processing Society (NAFIPS '98), Pensacola, FL, IEEE.

Ding, J., D. Berleant, et al. (2002). Mining MEDLINE: Abstracts, Sentences, or Phrases? Pacific Symposium on Biocomputing (PSB 2002), Kaua'i, Hawaii.

Duda, R. O. and P. E. Hart (1973). Pattern Classification and Scene Analysis. New York, Wiley.

Egan, M. A., M. Krishnamoorthy, et al. (1998). FCLUST: A Visualization Tool for Fuzzy Clustering. Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, Atlanta, Georgia.

Eisen, M. B., P. T. Spellman, et al. (1998). "Cluster analysis and display of genome-wide expression patterns." Proceedings National Academy of Science **95**: 14863-14868.

Fayyad, U., G. G. Grinstein, et al. (2002). Information Visualization in Data Mining and Knowledge Discovery. San Francisco, Morgan Kaufmann.

Fisherkeller, M. A., J. H. Friedman, et al. (1974). "PRIM-9: An Interactive Multidimensional Data Display and Analysis System." Technical Report SLAC-PUB-1408. Stanford Linear Accelerator Center. Stanford, CA.

Fukuda, K., T. Tsunoda, et al. (1998). Toward Information Extraction: Identifying Protein Names from Biological Papers. Proceedings of the Pacific Symposium on Biocomputing, Hawaii.

Gamma, E., R. Helm, et al. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Boston, Addison-Wesley.

Gregor, J. and M. G. Thomason (1993). "Dynamic Programming Alignment of Sequences Representing Cyclic Patterns." IEEE Transactions on Pattern Analysis and Machine Intelligence **15**(2): 129-135.

Hagiwara, M. (1992). Extended Fuzzy Cognitive Maps. 92 IEEE Int Conf Fuzzy Syst FUZZ-IEEE, San Diego, IEEE.

Hastie, T. and W. Stuetzle (1989). "Principal Curves." Journal of the American Statistical Association **84**(406): 502-516.

Hedden, P. and A. L. Phillips (2000). " Gibberellin metabolism: new insights revealed by the genes." Trends Plant Sci. **5**: 523-530.

Higuera, C. d. l. and F. Casacuberta (2000). "Topology of strings: Median string is NP-complete." Theoretical Computer Science **230**: 39-48.

Himberg, J. (1998). Enhancing SOM-based Data Visualization by Linking Different Data Projections. Proceedings of 1st International Symposium on Intelligent Data Engineering and Learning (IDEAL '98), Hong Kong.

Hoffman, P. E. and G. G. Grinstein (2002). A Survey of Visualizations for High-Dimensional Data Mining. Information Visualization in Data Mining and Knowledge Discovery. A. Wierse. San Francisco, Morgan Kaufmann.

Jain, A. K. and R. C. Dubes (1988). Algorithms for Clustering Data. Englewood Cliffs, NJ, Prentice-Hall.

Jiang, X., A. Munger, et al. (2001). "On Median Graphs: Properties, Algorithms, and Applications." IEEE Transactions on Pattern Analysis and Machine Intelligence **23**(10): 1144-1151.

Johnson, D. (1975). "Finding All the Elementary Circuits of a Directed Graph." SIAM Journal on Computing **4**(1): 77-84.

Kanehisa, M. and S. Goto (2000). "KEGG: Kyoto Encyclopedia of Genes and Genomes." Nucleic Acids Research **28**(1): 27-30.

Karp, P. D., M. Krummenacker, et al. (1999). "Integrated pathway/genome databases and their role in drug discovery." Trends in Biotechnology **17**(7): 275-281.

Karp, P. D., M. Riley, et al. (2000). "The EcoCyc and MetaCyc databases." Nucleic Acids Research **28**(1): 56-59.

Kaski, S., J. Venna, et al. (1999). Coloring that Reveals High-Dimensional Structures in Data. Proceedings of the 6th International Conference of Neural Information Processing.

Kinnear, K. E. (1994). Advances in genetic programming. Cambridge, Mass., MIT Press.

Kohonen, T. (1984). Self-Organization and Associative Memory. Berlin, Springer-Verlag.

Kohonen, T. (1985). "Median Strings." Pattern Recognition Letters **3**: 309-313.

Kohonen, T. (1995). Self-Organizing Maps. Berlin, Springer-Verlag.

Kohonen, T. (1997). Self-Organizing Maps. Berlin, Springer.

Kohonen, T. (2001). Self-Organizing Maps. Berlin, Springer.

Kosko, B. (1986). "Fuzzy Cognitive Maps." International Journal Man-Machine Studies **24**: 65-75.

Kosko, B. (1986). "Fuzzy Knowledge Combination." International Journal of Intelligent Systems **1**: 293-320.

Kosko, B. (1992). Neural Networks and Fuzzy Systems. Englewood Cliffs, Prentice Hall.

Koza, J. R. (1992). Genetic programming : on the programming of computers by means of natural selection. Cambridge, Mass., MIT Press.

Koza, J. R. (1994). Genetic programming II : automatic discovery of reusable programs. Cambridge, Mass., MIT Press.

Krishnapuram, R. and J. M. Keller (1994). Fuzzy and Probabilistic Clustering Methods for Computer Vision. Neural and Fuzzy Systems. W. Kraske. Bellingham, WA, SPIE Inst. **IS-12**.

Lee, K., S. Kim, et al. (1996). "On-line fault diagnosis by using fuzzy cognitive maps." IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **E79-A,**(6): 921-922.

Liang, S., S. Fuhrman, et al. (1998). REVEAL, A general reverse engineering algorithm for inference of genetic network architectures. Pacific Symposium on Biocomputing 3, Hawaii.

Maes, M. (1990). "On a Cyclic String-to-String Correction Problem." Information Processing Letters **35**: 73-78.

Martinez-Hinarejos, C. D., A. Juan, et al. (2000). Use of Median String for Classification. Proceedings of the 15th International Conference on Pattern Recognition.

Marzal, A. and S. Barrachina (2000). Speeding Up the Computation of the Edit Distance for Cyclic Strings. Proceedings of the 15th International Conference on Pattern Recognition, Barcelona, Spain.

Mateti, P. and N. Deo (1976). "On Algorithms for Enumerating all Circuits of a Graph." SIAM Journal on Computing **5**(1): 90-99.

MathWorks, T. (2002). Fuzzy Logic Toolbox. http://www.mathworks.com/products/fuzzylogic/index.shtml. Date accessed: June 18, 2002.

Matsuno, H., Doi, A., Nagasaki, M. and Miyano, S. (2000). Hybrid Petri Net Representation of Gene Regulatory Network. Pacific Symposium on Biocomputing 5, Hawaii.

McDonald, J. A. (1982). "Interactive Graphics for Data Analysis." Technical Report Orion II. Statistics Department, Stanford University.

Mollineda, R. A., E. Vidal, et al. (2000). Efficient Techniques for a Very Accurate Measurement of Dissimilarities between Cyclic Patterns. Lecture Notes in Computer Science. Berlin, Springer-Verlag. **1876:** 337-346.

Mulier, F. and V. Cherkassky (1995). "Self-Organization as an Iterative Kernel Smoothing Process." Neural Computation **7**: 1165-1177.

Nadler, M. and E. Smith (1993). Pattern Recognition Engineering. New York, Wiley, John & Sons.

Ng, S.-K. a. M. W. (1999). "Toward routine automatic pathway discovery from on-line scientific text abstracts." Genome Informatics **10**: 104-112.

O'Rourke, J. (1998). Computational Geometry in C. New York, Cambridge University Press.

Overbeek, R., N. Larsen, et al. (2000). "WIT: integrated system for high-throughput genome sequence analysis and metabolic reconstruction." Nucl. Acids. Res. **28**: 123-125.

Reisig, W. and G. Rozenberg (1998). Lectures on Petri Nets I: Basic Models. Berlin, Springer.

Ripley, B. D. (1996). Pattern Recognition and Neural Networks. Cambridge, Cambridge University Press.

Sammon, J. W. (1969). "A Nonlinear Mapping for Data Structure Analysis." IEEE Transactions on Computers **C-18**(5): 401-409.

Shatkay, H., S. Edwards, W. J. Wilbur and M. Boguski (2000). Genes, Themes and Microarrays. Int. Conf. on Intelligent Systems in Molecular Biology.

Stapley, B. J. a. G. B. (2000). Biobibliometrics: Information retrieval and visualization from co-occurrences of gene names in medline asbtracts. Pacific Symposium on Biocomputing 5, Hawaii.

Sun, T. (2000). " Gibberellin Signal Transduction." Curr. Opin. Plant Biol. **3**: 374-380.

Sutherland, P., A. Rossini, et al. (2000). "Orca: A Visualization Toolkit for High-Dimensional Data." Journal of Computational and Graphical Statistics **9**: 509-529.

Swayne, D. F., D. Cook, et al. (1998). "XGobi: Interactive Dynamic Graphics in the X Window System." Journal of Computational and Graphical Statistics **7**(1): 113-130.

Tarjan, R. (1972). "Depth-First Search and Linear Graph Algorithms." SIAM Journal on Computing **1**(2): 146-160.

Tarjan, R. (1973). "Enumeration of the Elementary Circuits of a Directed Graph." SIAM Journal on Computing **2**(3): 211-216.

Thearling, K., B. Becker, et al. (2002). Visualizing Data Mining Models. Information Visualization in Data Mining and Knowledge Discovery. A. Wierse. San Francisco, Morgan Kaufmann.

Tiernan, J. C. (1970). "An Efficient Search Algorithm to Find the Elementary Circuits of a Graph." Communications of the ACM **13**(23): 722-726.

Tomita, M. (2001). "Whole-cell simulation: a grand challenge of the 21st century." Trends Biotechnol **19**(6): 205-210.

Tomita, M., K. Hashimoto, et al. (1997). E-CELL: Software Environment for Whole Cell Simulation. Genome Inform Ser Workshop Genome Inform.

Tomita, M., K. Hashimoto, et al. (1999). "E-CELL: software environment for whole-cell simulation." Bioinformatics **15**(1): 72-84.

Usuzaka, S., K.L. Sim, M. Tanaka (1998). A machine learning approach to reducing the work of experts in article selection from database: a case study for regulatory relations of S. cerevisiae genes in MEDLINE. Ninth Workshop on Genome Informatics.

Vesanto, J. (1999). "SOM-Based Data Visualization Methods." Intelligent Data Analysis **3**: 111-126.

Wagner, R. A. and M. J. Fischer (1974). "The String-toString Correction Problem." Journal of the Association for Computing Machinery **21**(1): 168-173.

Weaver, D. C., C. T. Workman, et al. (1999). Modeling Regulatory Networks with Weight Matrices. Pacific Symposium on Biocomputing 4, Hawaii.

Weinblatt, H. (1972). "A New Search Algorithm for Finding the Simple Cycles of a Finite Directed Graph." Journal of the Association for Computing Machinery **19**(1): 43-56.

Wong, L. (2001). PIES, A Protein Interaction Extraction System. Pacific Symposium on Biocomputing, Hawaii.