5-2008

# Visualization of an Approach to Data Clustering

Marisabel Guevara
*University of Arkansas, Fayetteville*

# VISUALIZATION OF AN APPROACH TO DATA CLUSTERING

# VISUALIZATION OF AN APPROACH TO DATA CLUSTERING

A thesis submitted in partial
fulfillment of the requirements of
Honors Studies in Computer Engineering

By

Marisabel Guevara

May 2008
University of Arkansas

## ABSTRACT

Using visualization and clustering goals as guidelines, this thesis explores a graphic implementation of a data clustering technique that repositions vertices by applying physical laws of charges and springs to the components of the graph. The resulting visualizations are evidence of the success of the approach as well as of the data sets that lend themselves to a clustering routine. Due to the visual product of the implementation, the algorithm is most useful as an aid in understanding the grouping pattern of a data set. Either for a rapid analysis or to assist in presentation, the visual result of the clustering approach is a useful tool for discovering trends in a data set.

This thesis is approved for recommendation to the Honors College.

Thesis Director:

______________________________________
Dr. James P. Parkerson

Thesis Committee:

______________________________________
Dr. Russell Deaton

______________________________________
Dr. Craig Thompson

## ACKNOWLEDGEMENTS

Thank you Dr. Thomas Schweiger for being my mentor beyond the scope of an internship – your commitment to learning has been the best lesson. Thank you Dr. James P. Parkerson and Dr. Craig Thompson for continuous guidance and advice.

I also thank my family for giving me opportunities and support despite great physical distance, and my friends for making all memories valuable in their own way.

## TABLE OF CONTENTS

## LIST OF FIGURES

# 1. INTRODUCTION

Clustering data is a field of data mining where meaningful information can be extracted from a data set by identifying and exploiting common traits between the data elements. The benefits of successful classification of components within data sets are best illustrated by the multiple applications for data clustering visible throughout nature and society. Ecological studies, for example, require ordination of climate, food, or migration data in order to identify life structures such as biospheres or ecosystems. Researchers in ecology utilize computing tools to solve such problems, yet selecting an algorithm that fits the characteristics of ecological data is a challenge [3]. Other examples of fields that use classifying routines as a fundamental tool to organize data are medicine in assembling reports such as electrocardiograms, market research to identify potential consumer segments from surveys or test panels, or internet search result providers that enhance their results through intelligent grouping. There is not only a need for clustering techniques, but also an overwhelming quantity of data to be processed by these techniques.

The majority of applications of information clustering, including all of the above examples, very often benefit when represented visually in an effective way. The goal of visualization is to emphasize not the data but the underlying phenomenon. It is estimated that fifty percent of the human brain's neurons are associated with processing the large bandwidth of information received through vision [8]. Thus, not only must visualization provide an accurate representation of a data set, but this representation will be interpreted at a very high speed. Acknowledging the capacity and rapidity of human interpretation means that the visualization must at all times have the desired effect in order to fulfill the primary goal of visualization. In a sense, the development of a visualization technique is related to that of advertisements. In the case of a single static frame, for example, the initial impact of the visual display is most significant. Although this seems contradictory since there are graphs that require close scrutiny of the legend, axes labels, and units of measurement that are used by professionals across multiple fields, this family of graphs does not qualify as visualization. The difference lies in that the value of visualization is in extracting the meaning of information, and is not the same as a graphical representation that serves as an alternative to a table or a similar data structure for expressing results. Using this terminology, an example of graphical representation of data is a set of temperature measurements plotted as points relative to an axis that marks the distance from a heat source. A possible visualization of this same data would be a heat distribution pattern that utilizes color coding for the different temperatures and the test area itself provides the shape of the image. Visualization uses graphical primitives, such as coordinate planes, and exploits visual techniques and display technology to provide insight into the information presented.

The research of this thesis explores the ability to enhance visualization of data sets that exhibit partitioned subsets. The display is a two dimensional graph of points that represent each data element and edges that connect two points that have a relationship. The data sets used throughout this thesis to demonstrate the visualization results are a couple of social networking sources and the 2007 NCAA College Football matches. The sets of data have different types of relationships that connect their elements, and it must be noted that the presence of a relationship between two elements is provided to the clustering routine as it is not the goal of the algorithm to try to extract relationships from a given set of data. Thus, for the above mentioned data sets, the presence of a relationship between two points (i.e., an edge between two nodes) was determined by a subjective definition specific to the data set. This is a strength of the visualization technique because it is flexible to different applications and different attributes of the data can be emphasized depending on the definition of a relationship.

## 1.1 Problem

The visualization technique detailed in this thesis is meant to enhance the appearance of grouping trends in a data set. Visualization is as valuable as the insight it provides the viewer into a characteristic of the data that is not readily perceived. This thesis attempts to improve on a data set that is viewed as a two dimensional graph. Such a graph can be termed a network that contains elements, called vertices, connected by edges. Current clustering techniques attempt to group the vertices of a network into a set of subnetworks such that once a vertex belongs to one cluster it cannot belong to another. However, data sets resulting from scientific measurements or social analysis have anomalies, elements that follow no trend. For example, a set of vertices where each represents a middle school student and an edge connects two students if they are friends can be broken up into cliques by a clustering routine. There might be vertices that have one or zero connections, termed outliers, and there also exists a possibility of vertices that are connected to two or more cliques, also called hubs. By forcing all vertices to belong to one subgraph, the outliers are assumed to have as many associations to a clique as all its other members, and the multiple connections that a hub has to another subgraph are ignored. In the specific example of the middle school network, the ability to track how a rumor was distributed between the students, or maybe even something as important as the contamination track of a virus, would be lost on the clustered network. This example illustrates the impact of not being able to visualize all properties of partitioned data sets via existing clustering routines.

In a more specific setting, the implementation of this thesis is part of a research project at Acxiom Corporation. The lack of a strong visualization technique limits the effect of a presentation, and this was the case of the Network Properties application that the grouping routine detailed in this document is a part of. Some of the implications of the application's functionality, explained in further detail in Section 3.2, are not as apparent when the subgraphs of the displayed network are mixed together. The goal of this routine is thus to reposition vertices so that the clusters, outliers, and hubs in the network are more evident.

## 1.2 Thesis Statement

Provided a data set with defined relationships between its elements, visualization of the data is enhanced with the use of a routine that positions the data nodes on a two dimensional coordinate plane emulating the physical behavior of electric point charges connected by springs.

## 1.3 Approach

Abstracting a data set into a graph of vertices and edges, as is the case of the environment that this clustering visualization must exist within, is a good means to guarantee a routine that can be applied to different types of data. The repositioning of the vertices of the network must thus be based on its graphical elements rather than specific properties of the data set. This method uses the vertex and edge information of the network on modified physics equations ordinarily used to model the behavior of electric charges and springs. Each vertex is represented as an electric charge in the algorithm, and the presence of an edge between two vertices is fashioned to act as a spring between the two electric charges. This technique for relocating graphical points to achieve a visualization of the source data set does not distinguish between two types of charges. Instead, all elements of the data set are represented by the same type of electric charge and will always repel, as explained in Section 2.1.1. Considering the graphical representation of the data, it does not make sense to have two different types of points as this distinction cannot be expressed in the graph itself. Another key aspect of the approach is that only one spring can exist between two charges, meaning one edge between two vertices. In terms of the data set, this means that either a relationship exists or does not exist between two elements. A network thus becomes a set of moving points connected by springs, and eventually all the forces will reach a state where an equal and opposing force balances the system, designated as the point of equilibrium.

## 1.4 Potential Impact

The approach taken by this research to the problem of visualizing data sets that have partitions can be applied to the analysis of trends in society that have not been easily spotted before. The ability to rapidly identify a hub, for example, that is offsetting data patterns cannot be achieved by a simple comparison of the degrees of vertices or the clustering technologies reviewed in Section 1.1. The hub can be of much importance to society in the case of analyzing the spread of a disease, or linking groups suspected of delinquent activity. The flexibility of defining the edges as any relationship without changing the functionality of the routine is also significant. The challenge of modifying an algorithm to suit a specific science's use is lessened by this technique. In the scope of Acxiom Corporation's products, this tool could assist the communications with a client. It could even aid in selling the services that Acxiom offers by allowing the potential customer to better visualize the existing problem with naming and addressing gaps in society.

## 1.5 Organization of this Thesis

Chapter 2 of this thesis reviews the physical concepts and equations that are utilized by this clustering technique, the details of the graphic components of Java that affected the implementation, and literature of related work in the area of clustering. The details of the approach to this visualization, including an overview of the application that serves as the environment and a walk-through of the algorithm, are described in Chapter 3. The following chapter describes the data structures used in Java as well as how the physics equations were applied to the graphical elements of a network. Chapter 5 presents an analysis of the resulting visualization on different data sets, and Chapter 6 concludes this thesis with the contributions of this work and insight into possible extensions to the implementation.

# 2. BACKGROUND

## 2.1 Key Concepts

The routine that enhances the visual effect of graphed data elements is derived from the physical properties of electric charges and springs. This behavior is quantified by Coulomb's Law and Hooke's Law, and reviewing the basics of each is fundamental to understanding the reason for this visualization method's success.

Implementing the visual part of the routine in Java required the use of a JLabel object customized for the specific task of displaying vertices and edges. Tailoring an object in Java is achieved by adding necessary data structures and methods that are specific to the application's needs. The extensions made to the standard JLabel object in this implementation are not important to the topic of this thesis. The functionality of the repaint() method of a JLabel, however, does affect the visualization result and the details are explained in Section 2.1.2.

### 2.1.1 Coloumb's Law

An atom is comprised of electrically charged particles called protons and electrons along with a set of particles called neutrons that have no charge at all. Conventionally the protons are said to have a positive charge and the electrons have a negative charge, yet the concept of positive and negative is arbitrary and not utilized in this thesis. The property of electrically charged sub-atomic particles that is essential, however, is that particles with the same charges repel each other and particles with opposite charges attract each other [7]. A force that either pushes or pulls is generated between any two charges and if the charges are free to move then they will accelerate apart or together. The magnitude of the force determines the amount of the movement, and Coulomb's law states that the force between two electric point charges is proportional to the magnitude of the charges and inversely proportional to the square of the distance between the charges. The equation to calculate the size of this force is:

$$F_{A\text{-}B} = k_c \times \frac{(q_A \times q_B)}{r^2}$$

| | |
|---|---|
| $F_{A\text{-}B}$ | => force between charges A and B |
| $k_c$ | => Coulomb's constant, with the value $9\times10^9$ $N*m^2/C^2$ |
| $q_A$, $q_B$ | => magnitudes of electric charge A and B respectively, in Coulomb units |
| $r$ | => distance between point charge A and point charge B |

For a two dimensional application the force must be separated into a vertical and a horizontal component to graph the movement on a Cartesian coordinate system. The Euclidean distance between two points is given by a formula derived from the Pythagorean Theorem. The relationship between the distance of points A and B on a xy plane is expressed in the following equation:

$$d = \sqrt{\,[(x_A - x_B)^2 + (y_A - y_B)^2]}$$

| | |
|---|---|
| $d$ | => distance between two points on a xy plane |
| $x_A$, $x_B$ | => x coordinates of A and B respectively |
| $y_A$, $y_B$ | => y coordinates of A and B respectively |

Finally, in order to extract the horizontal and vertical components of the force between two electric point charges it is necessary to use trigonometric ratios of angles [7]:

$$F_x = F_{A\text{-}B} \times \cos\Theta \qquad F_y = F_{A\text{-}B} \times \sin\Theta$$

| | |
|---|---|
| $\Theta$ | => angle formed by the vector of force F and its horizontal or vertical component |

$$\cos\Theta = \frac{(x_A - x_B)}{d} \qquad \sin\Theta = \frac{(y_A - y_B)}{d}$$

Combining Coulomb's Law applied to a Cartesian coordinate system and the Pythagorean Theorem allows an extraction of the horizontal and vertical components of the force between two electric point charges:

$$d = \sqrt{[(x_A - x_B)^2 + (y_A - y_B)^2]}$$

$$F_x = \frac{k_c \times (q_A \times q_B)}{d^2} \times \frac{(x_A - x_B)}{d}$$

$$F_y = \frac{k_c \times (q_A \times q_B)}{d^2} \times \frac{(y_A - y_B)}{d}$$

This equation only provides the magnitude of the force without direction, and in this thesis the task of determining the direction is simplified by the fact that the implementation's abstraction of the data elements as electric charges assumes all points have the same charge. Since like charges repel, then the resulting forces between the points are always directed away from each other.

### 2.1.2 Hooke's Law

A spring will return to its original shape after being stretched because it is an elastic object. This property is due to a force, called a restoring force[9], that pulls the spring back to its initial state whenever it is stretched. Hooke's Law quantifies the relationship between the restoring force and the distance the spring has been stretched:

$$F = -k_s \times d$$

| | |
|---|---|
| $F$ | => restoring force |
| $k_s$ | => spring constant |
| $d$ | => displacement by stretching |

The displacement from point A to point B expressed in horizontal and vertical components for a Cartesian coordinate system can be derived from trigonometric properties:

$$\Delta x = \frac{x_A - x_B}{d} \qquad \Delta y = \frac{y_A - y_B}{d}$$

| | |
|---|---|
| $x_A$, $x_B$ | => x coordinates of A and B respectively |
| $y_A$, $y_B$ | => y coordinates of A and B respectively |
| $d$ | => Euclidean distance between A and B |

The resulting equations for the horizontal and vertical components of the restoring force are:

$$F_x = -k_s \times d \times \frac{(x_A - x_B)}{d} \qquad F_y = -k_s \times d \times \frac{(y_A - y_B)}{d}$$

$$F_x = -k_s \times (x_A - x_B) \qquad F_y = -k_s \times (y_A - y_B)$$

### 2.1.3 Threads of Execution

The JLabel class in Java inherits paint() and repaint() methods from the JComponent interface which is a part of the Swing framework that assists with designing graphical user interfaces (GUI). The JLabel is an element that serves as the part of a GUI that displays text, an image, or both [12]. The execution of an application with a GUI typically starts with an intialization, followed by some processing, then the GUI is updated, and these last two steps are repeated as many times as necessary until the program is terminated. In the case of a Java application using a JLabel as the display area, the repaint() method is invoked in order to update the text and/or image on the display area. When repaint() is called, the JLabel's paint() method is executed after all the pending events have been completed[12].

A sample usage of a JLabel is a simple display that draws a stick to keep tally of every Z it finds in a document as the program reads the text. There is no input from the user to the application, and the processing step between each repaint() is the actual reading of the document. The pseudocode below is a possible implementation of the example:

```
while( !EOF ){
    char = next character
    if ( char == 'Z' ) {
          tally++
          repaint()
    }
}
```

In the pseudocode, the variable tally is the total number of Zs found in the document and is accessed by the paint() method of the display component to determine the number of sticks to draw. Assuming a document large enough that the program requires several seconds to scan each character, the expected execution of the program would start with a blank display area that with time would accumulate more sticks. The assumption of the document size is to establish a scenario where the computation takes enough time that the visual aspect of it can be appreciated by humans. The actual execution, however, would display a blank screen for almost the entire time of the program's execution and right before terminating would display all the sticks. This final screen might even be displayed such a brief amount of time that a human viewing the execution would not recognize that it, in fact, did appear.

The unexpected execution is due to a characteristic of the execution method of Swing components. As the explanation of the repaint() method's functionality states, "The component will be repainted after all of the currently pending events have been dispatched"[12]. When executing the pseudocode example, the Java Runtime

Environment optimizes the repainting of the display by grouping all the calls of repaint() together. The file I/O and other processing that needs to be done will thus execute faster, but the desired functionality is not achieved. Java suggests a convention for utilizing Swing components known as the Swing Single Threading Rule which states “To avoid the possibility of deadlock, you must take extreme care that Swing components and models are created, modified, and queried only from the event-dispatching thread”[4]. Since Swing components are not build to be thread safe, this warning is necessary before a programmer attempts to utilize threads other than the event-dispatching thread in an application with Swing components. However, there are a few methods that are exempt from this rule, and repaint() is included in the list of methods excluded from the Swing Single Threading Rule [4].

Java includes a Thread class that can be instanciated in an application when an asynchronous task will be performed. An instance of the Thread class is a Thread object and the execution tasks that the Thread object will perform are detailed in its run() method:

```
public void run() {
        System.out.println("Hello from a thread!");
}
```

The pseudocode for the Z-counting example above can be modified to utilize a new thread to execute the repaint() while the event-dispatching thread handles the file I/O:

```
while( !EOF ){
    char = next character
    if ( char == ‘Z’ ) {
          tally++
          //Create a new thread for repainting
          Thread repainter = new Thread() {
                public void run() {
                      repaint()
                }
}
//Run repainter thread
repainter.start()
    }
}
```

This new pseudocode utilizes Java’s built in tools for the specific task of threading a Swing component, and this concept becomes central to the implementation of the visualization routine.

## 2.2 Literature Review

The use of physics equations to move the vertices of a graph as a clustering technique is generally known as a force-based or spring algorithm [16]. A force-based algorithm is any algorithm that assigns forces to the elements of a graph using Coulomb’s Law and Hooke’s Law. Andreas Noack reviews different force-based models in his paper “Energy Models for Graph Clustering” [10] and differentiates between the purpose of a model and that of an algorithm. Xiaowei Xu, Nurcan Yuruk, and Thomas Schweiger propose an algorithm for clustering that innovates through its definition of the problem in “SCAN: A Structural Clustering Algorithm for Networks”[15]. This paper introduces the contribution of neighborhoods to graph partitioning, a topic that was also of interest in

2003 as shown by Bernd Wiswedel, David E. Patterson, and Michael R. Berthold in "Interactive Exploration of Fuzzy Clusters Using Neighborgrams"[14], which pertained to their research of a one-dimensional visualization of a clustering technique. Kang Zhang further explores the graphic guidelines for information visualization in "From Abstract Painting to Information Visualization". These papers contributed to the goals and definitions of this thesis as they track the interest and progress in graph partitioning and visualization.

"Energy Models for Graph Clustering" evaluates energy models based on how well a graph layout reflects the clusters within that graph. Most energy models were designed for visualization rather than partitioning a graph, thus their primary purpose was to provide a layout with short and uniform edge lengths where the nodes are evenly distributed. The author presents two definitions that make up an energy-based graph layout method: an energy model is in charge of describing the layout to be created, while an energy minimization algorithm stipulates the actual computations to reach the desired layout. The author presents two energy models: "node-repulsion LinLog", where the nodes experience forces between each other like electric charges in Coulomb's Law, and "edge-repulsion LinLog", a less common model where the edges are pushed and pulled away from each other. Both models aim for internal validity, where the nodes that are more densely connected are grouped and the ones with fewer connections are separated, as opposed to external validity in which grouping depends on an external definition of a cluster. Both "node-repulsion LinLog" and "edge-repulsion LinLog" use edge weights as an important part of the model, a significant difference to the approach taken by this thesis that clusters graphs with undirected edges of the same weight. However, the results for the two models explored in “Energy Models for Graph Clustering” are interesting and have a strong relation to the goals of this thesis. The first energy model, the “node-repulsion LinLog,” builds graphs where the clusters are strongly separated from each other by a large distance, and the edge lengths within a cluster are much smaller. This is an accomplishment of internal validity, yet the second model does not create layouts with such strong distinctions between clusters. The “edge-repulsion LinLog” model provides a symmetric layout with overall short edge lengths, and hence it does not relate as strongly to this thesis despite the innovative idea of reversing the force elements in a graph.

SCAN, the algorithm detailed in “SCAN: A Structural Clustering Algorithm for Networks,” is able to detect the clusters, hubs, and outliers of a graph. Contrary to typical clustering techniques that seek to define the membership of every vertex in the graph, SCAN differentiates between the roles that different vertices play in a network. This sets SCAN’s goals along the same track as the objectives of this thesis, and it is also evidence that recent development to identify more than only the subgraphs of a network has taken place. Not only is the end result of SCAN unique, but the approach is also different from those of traditional algorithms. Whereas the most common clustering procedure is to seek a large number of edges within elements that belong to a cluster and a smaller number of edges between clusters, SCAN focuses on the neighborhood of each cluster as the criteria for grouping. SCAN groups vertices according to how they share neighbors, creating partitions that are termed structure-connected clusters. Each vertex is visited once for the algorithm to determine the structure-connected clusters, and then it visits the vertices that have been isolated from any cluster to identify these as hubs or outliers. This focus on the value of a neighborhood dates back to the research of an article titled “Interactive Exploration of Fuzzy Clusters Using Neighborgrams.” This article attempts to solve the clustering of a graph problem without the previous knowledge of how many subgraphs exist, which was the common approach to graph partitioning when the article was written in 2003. It proposes a routine that first analyzes the neighborhood of each vertex and next selects one vertex from that neighborhood as the optimal cluster representative. Thus, a “neighborgram” is generated for each pattern in the data set and then a nearest-distance to the cluster representative approach finishes grouping the vertices that are not clearly members of one cluster. The article relates to this thesis because it identifies that the clustering of a graph is actually separating a data set by a trend, or pattern, that is not explicitly detailed in the graph itself. This is the goal of visualization, to express a hidden phenomenon of the data, and the one-dimensional graphic representation provided in “Interactive Exploration of Fuzzy Clusters Using Neighborgrams” is a beginning to exploring information visualization.

An article that specifically explores the importance of visualizing data is “From Abstract Painting to Information Visualization,” which compares techniques of abstract painting with information visualization. It introduces the concept of aesthetic computing, a name for applying artistic goals to computation. The author believes that grouping data is “one of the most important processes in visual data mining and information visualization” [16], and for this purpose outlines five factors that determine grouping in a visual context. The first is proximity, as elements appear grouped together if they are near each other. Similarity is grouping items together that are similar by some measure. The third factor, closure, groups elements together if they tend to complete some entity.

Continuity refers to the perception that items lying on a line belong to the same group. Finally, simplicity is the term used by the author to describe grouping points together into simple figures, such as circles, triangles, or squares. These "Laws of Organization" [16] are important in the field of abstract art and can be extended to visual data mining such that symmetry and minimal crossing in a graph allow a useful representation to also be visually appealing. These definitions assisted in understanding how each part of the resulting graph contributes to the viewer's understanding of the data. The articles all played a part in understanding the objective of this thesis and keeping it in line with the current development in data mining.

# 3. ARCHITECTURE

## 3.1 High Level Design

The visualization is accessible as part of a Java application that includes other tools for manipulating the data being displayed. The Java application is a preexisting environment for the visualization because this research is a contribution to a larger project. Therefore, the other components of the application will be discussed briefly but are not the focus of this thesis. A screenshot of the Java application is available in Figure 1, and the functionality of each section of the GUI is presented in Section 3.2.

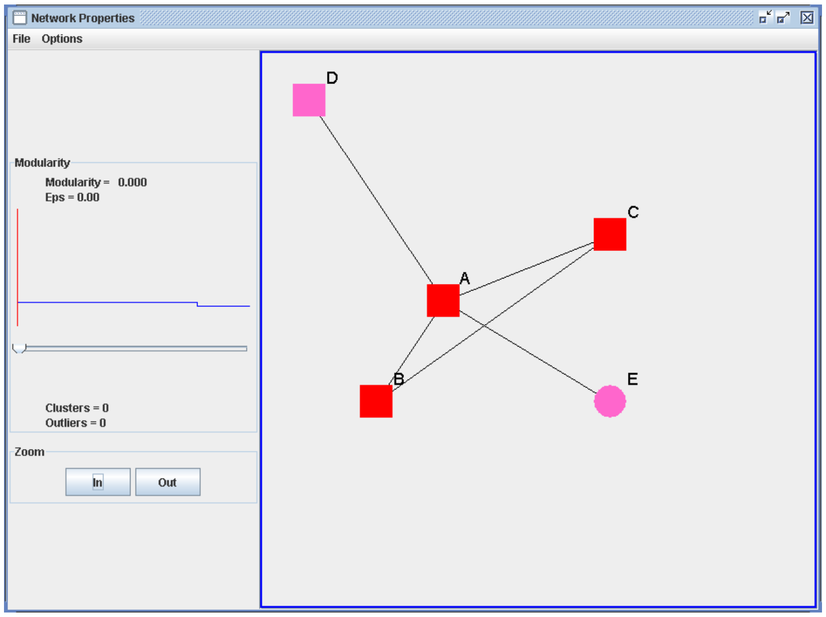


Figure 1: Screenshot of Network Properties Application

The single design consideration that was a result of the approach itself rather than the setting of the implementation was the need for a limit in the number of times the vertices would be repositioned. In the optimization routine, each data element is represented as an electric charge in two dimensional space and the relationships between the data elements are the springs connecting the electric charges. The behavior of electric charges in space as defined by physical forces is detailed in Section 2.1.1. When viewed from the graphical representation of the data sets in the application's GUI, the vertices are electric charges of the same type and the edges are the springs for the optimization routine. By this abstraction, the vertices will repel each other and only the presence of a spring between two point charges will keep them close together. Eventually the set of charges and springs will reach a point of equilibrium where all forces are opposed by forces of equivalent magnitude. A more accurate definition of the equilibrium is actually a state where all forces are opposed by forces of almost equivalent magnitude. The optimal finish point for the routine is arrived at when the equilibrium state is met, yet the possibility for a case of vertices repeatedly moving back and forth an insignificant amount exists and must be accounted for.

Additional design restrictions for the visualization were due to the predefined application. The setting is a two dimensional layout of vertices connected by points, and both vertices and points are drawn on the GUI with classes that are part of Java's Swing components. The algorithm can be initiated by selecting "Optimize" via the "Options" drop down menu located directly underneath the application's titlebar at the top of the window, as displayed in Figure 2. Section 2.1.3 details both the functionality of a Swing component's painting method and the difficulty of combining Swing component painting with any other type of processing. Due to this unexpected behavior of Swing components, the threading solution that is introduced in Section 2.1.3 had an effect on the visualization implementation.

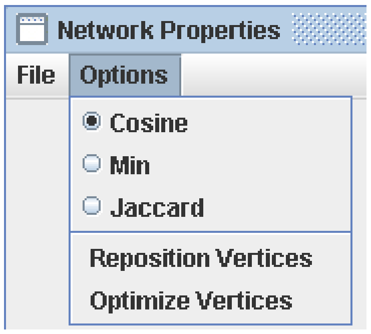


Figure 2: Options Menu of the Network Properties Application

## 3.2 Components of the GUI

The GUI is made up of a menu bar, a sidebar, and a large display area that shows the message "(No Network Loaded)" when the program is started. This area is the section where the graphical representation of a loaded network will be displayed. The "File" menu available on the menu bar holds the "Open" and "Save" options that can be selected to either load or store a file. When either is chosen, a file navigator window opens which allows the user to explore available files and select either a GraphML or a DAT file, as seen in Figure 3.

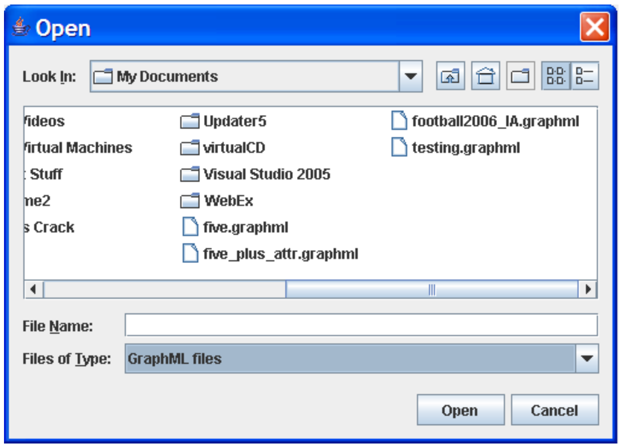


Figure 3: File Navigator of the Network Properties Application

The second menu available on the menu bar is named “Options” and it allows the user to select one of three alternatives labeled “Cosine”, “Min”, and “Jaccard”. The default is “Cosine” and the use of radio buttons designates which of the three options is active, as seen in Figure 2. The “Options” menu also includes two more selections named “Reposition Vertices” and “Optimize Vertices.” Both options have the same goal of visually grouping any subgroups present in the displayed network but are implemented differently. “Reposition Vertices” has a longer execution time than “Optimize Vertices” and does not keep the graphical representation of the network updated throughout its processing time. “Optimize Vertices” executes the visualization routine detailed in this thesis.

The sidebar of the GUI holds two components, “Modularity” and “Zoom”, the second underneath the first. The “Modularity” element is used to display a small graph of modularity versus similiarity, a sample of which is shown in Figure 4. For this application, similarity is a measure of how strongly two vertices are related as defined by a specific relationship given to each edge in the network. The modularity is a quantification of how modular the network is when all edges that have a similarity less than a given value are ignored. Thus, the slider bar placed underneath the small graph in the “Modularity” component of the GUI allows the user to set a specific similarity and the graph of the network is redrawn to exclude edges that have a similiarity less than the one specified by the slider. The “Zoom” tools are two buttons, “In” and “Out”, that allow the user to alter the magnification on the network display.

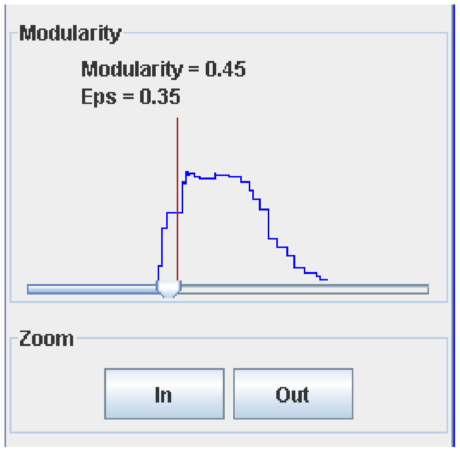


Figure 4: Modularity Component in Network Properties Application

## 3.3 Optimize Vertices

Selecting Option→Optimize Vertices on the running application executes some necessary steps in addition to the actual calculation of forces on each vertex and relocating the vertices based on the computations. Before calculating the forces it is necessary to verify that no two vertices have the same location. If two vertices were to be located in exactly the same position, then calculating the Euclidean distance, explained in Section 2.1.1, between the two vertices would result in a zero. The force exerted on a vertex by another vertex is inversely proportional to the square of the distance between the two vertices, and if the distance was exactly zero then a divide by zero would occur. An attempt to divide by zero in a computer program results in undefined behavior, therefore it is essential to assure that this will not occur by relocating the vertices prior to calculating the forces on each vertex.

After the vertices have been relocated, the concept of an end point to the repositioning must be defined. As mentioned in Section 3.1, all forces will theoretically reach equilibrium such that each becomes negligible due to other equally opposing forces in the network. Since the forces result in the moving of a vertex, keeping track of the largest move made in the most recent set of relocations allows a limit to be set. Thus, a previous move must have been at least the size of the limit in order to iterate through the calculations one more time. Finally, before each new repainting of the graphical display, the entire graph needs to be shifted to locate as much of it as possible in the viewable display area. This can be done by anchoring the vertex with the smallest x and y coordinates to the top left corner of the display and adjusting the coordinates of all other vertices relative to the anchor. The graph can then be updated and the process of calculating forces, moving the vertices, adjusting them for the display, and repainting is repeated until the largest move is not big enough to start a new iteration.

# 4. IMPLEMENTATION

## 4.1 Data Structures

The data structures that hold the vertices and edges of the network that represent the elements and relationships of a data set are a part of the application that was predetermined but served the routine well. The vertices and edges are collected into two TreeMap structures:

```
Private TreeMap<String, Vertex> vertexMap;
Private TreeMap<Pair, Edge> edgeMap;
```

A TreeMap is a Java class where an element is identified by a specific key and all elements are sorted by their corresponding key. The TreeMap class guarantees O(log(n)) time for each addition, retrieval, or removal from the map[12]. The Vertex class is a custom class for this implementation that, amongst other things, stores a string identifier for the vertex, a set of vertex identifiers that make up the neighborhood of the vertex, and x and y locations. The Edge class is also an original class that, amongst other things, stores the string identifiers of the two vertices connected by the edge. The Pair class used as the key to any edge in the edgeMap is a third class that is not part of the standard Java collection of objects and it holds two strings, which in this implementation serve as unique identifiers for an edge.

Since the Vertex class holds the current location in terms of x and y of every member of the vertexMap, it is simple to iterate through the vertexMap and perform calculations using the current position as operands. The movement of a vertex will be defined by all the forces exerted on it. Therefore, for the force of each vertex in a network, its force relative to every other vertex must be calculated and summed to obtain a total force. This can be done by iterating through the vertices only once and calculating all the forces rather than iterating through all the vertices to calculate the force on each vertex. A single iteration through all the vertices also takes advantage of the fact that the force vertexA experiences from vertexB is equal and opposite to the force vertexA exerts on vertexB. Using the notation for force, if $F_{A\text{-}B}$ is the force exerted on vertexA by vertexB, then $F_{B\text{-}A}$ is the inverse of $F_{A\text{-}B}$. This reduces the number of calculations by half, yet it is necessary to create a data structure to store all the calculated forces. A two dimensional array allows a single value to be addressed by two keys, and so an array of size $\text{number}_{\text{VERTICES}}$ by $\text{number}_{\text{VERTICES}}$ can store $F_{A\text{-}B}$ at position array[a][b] and at the same time array[b][a] will hold $(-1)\times F_{A\text{-}B}$. The algorithm thus uses a two dimensional array for horizontal forces and a two dimensional array for vertical forces. Once all the forces are calculated, a vertex's horizontal movement can be determined by adding one row of the horizontal force array:

$$x_A = \Sigma(\text{array}[A][i]), \text{ where } \{i: 0 <= i < \text{number}_{\text{VERTICES}}\}$$

With the data structures in place, the final consideration for the implementation of this visualization routine is the use of the processing techniques to generate an improved graphical representation of the network.

## 4.2 Computing Forces in the Network

Applying the physical forces of Coulomb's law and Hooke's Law to the network is a matter of utilizing the data structures and implementing the equations. The physical properties of electric charges and springs are utilized in the network of vertices and edges by pushing all vertices apart as if they were charges of the same type and only the presence of an edge between two vertices will keep them close together. Hence, the system is made up of two opposing forces: the repelling force between vertices and the restorative force exerted by the presence of an edge. Coulomb's law quantifies the force between two point charges, as detailed in Section 2.1.1, and when applied to a two dimensional coordinate system the resulting equations are:

$$d = \sqrt{[(x_A - x_B)^2 + (y_A - y_B)^2]}$$

$$F_x = \frac{k_c \times (q_A \times q_B)}{d^2} \times \frac{(x_A - x_B)}{d}$$

$$F_y = \frac{k_c \times (q_A \times q_B)}{d^2} \times \frac{(y_A - y_B)}{d}$$

| | | |
|---|---|---|
| $d$ | => | Euclidean distance between points A and B |
| $k_c$ | => | Coulomb's constant |
| $x_A$, $x_B$ | => | x coordinates of A and B respectively |
| $y_A$, $y_B$ | => | y coordinates of A and B respectively |

This implementation of Coulomb's law assumes that all vertices will repel, and so $q_A$ and $q_B$ can be set to constant ones since the direction of the forces will be determined by the position of the vertices[6], resulting in:

$$F_x = \frac{k_c \times (x_A - x_B)}{d^3} \qquad F_y = \frac{k_c \times (y_A - y_B)}{d^3}$$

These are the equations for the horizontal and vertical components of the repelling force, and to obtain the opposing restoring force Hooke's law can be applied. It provides the relationship between the restoring force's horizontal and vertical components and the displacement of the spring being stretched, as detailed in section 2.1.2:

$$F_x = -k_s \times (x_A - x_B) \qquad F_y = -k_s \times (y_A - y_B)$$

| | | |
|---|---|---|
| $k_s$ | => | spring constant |
| $d$ | => | displacement of a spring by stretching |

In the implementation, each vertex experiences a repelling force from all other vertices and a restoring force from all edges that it is an end of. Therefore, the total force can be expressed in its horizontal and vertical components as:

$$F_{X\ A\text{-}i} = \Sigma_{i \neq A}( Q_{X\ A,i} + S_{X\ A,i} \times \delta_{(A,i)} )$$

$$\text{Where } Q_{X\ A,i} = \frac{k_C \times (x_A - x_B)}{d^3} \quad ,$$

$$S_{X\ A,i} = -k_S \times (x_A - x_B) \quad , \text{ and}$$

$$\delta_{(A,i)} = 0 \text{ if no edge, } 1 \text{ if edge}$$

$$F_{Y\ A\text{-}i} = \Sigma_{i \neq A}( Q_{Y\ A,i} + S_{Y\ A,i} \times \delta_{(A,i)} )$$

$$\text{Where } Q_{Y\ A,i} = \frac{k_C \times (y_A - y_B)}{d^3} \quad ,$$

$$S_{y\ A,i} = -k_S \times (y_A - y_B) \quad , \text{ and}$$

$$\delta_{(A,i)} = 0 \text{ if no edge, } 1 \text{ if edge}$$

Coulomb’s constant is rounded to the value of one hundred thousand because this is not an actual representation of electric charges, but rather a routine inspired by the relationship of force on a vertex and the distance from that vertex to all others. Thus, the force exerted on vertexA by vertexB can be calculated entirely from the x and y coordinates of the vertices.

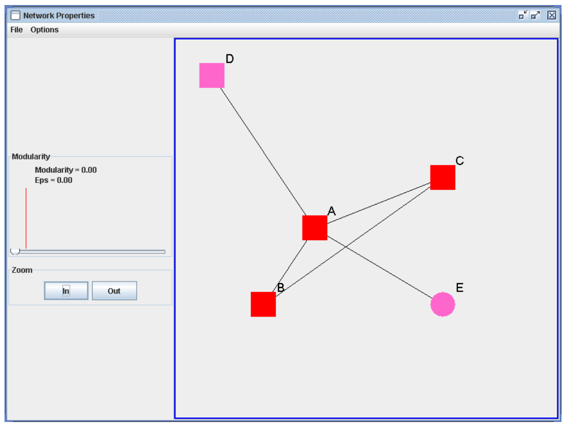
Once it is clear how the calculations will occur, piecing the entire design together is a matter of starting at the outermost layer of computation. The first step is to relocate all the vertices to different random positions, as described in Section 3.3. Next, the calculations will be performed so a new thread is created for the repainting of the display, which is detailed in Section 2.1.3. A loop is started that first relocates the vertices and then shifts the graph into the viewable area of the display. The display is repainted using the new thread, and the size of the largest move is tested against a set limit before commencing the iteration once again. The pseudo code below summarizes the algorithm:

```
Optimize( G=<V,E> ) {
    //Randomize vertices
    for each vertex vεV {
          v.setXcoordinate = Math.random();
          v.setYcoordinate = Math.random();
    }

    //Create new thread
    Thread repainter = new Thread() {
          //Detail functionality in run method
          public void run() {
                networkDisplay.repaint();
          }
    }

    //Optimize
    do {
          //Force calculations
          for each vertex v1εV {
                for each vertex v2εV {
                      calculate repelling force;
                      if( v1 and v2 are neighbors ) {
                            subtract restoring force;
                      }
                      store force in 2D array;
                }
          }

          //Vertex relocation
          for each vertex vεV {
                sum total force on v from 2D array;
                v.setXcoordinate = ratio * totalXforce;
                v.setYcoordinate = ratio * totalYforce;
                max = size of largest move;
          }

          //Shift to viewable area
          find lowest x and y;

          //Repaint
          repainter.start();
    } while( max > limit );
}
```

# 5. ANALYSIS

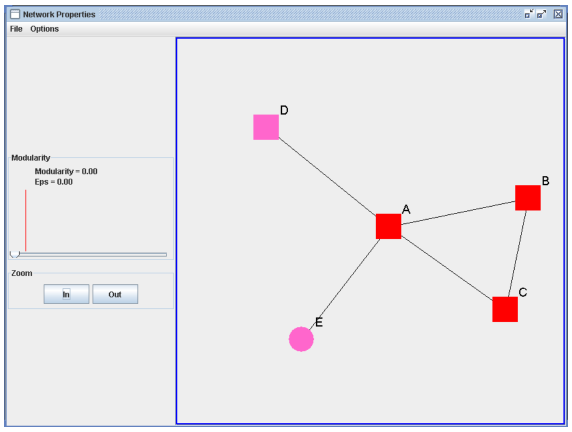
## 5.1 Small Example

For the purpose of demonstrating the repositioning of vertices performed by the algorithm, a small example graph with five vertices is depicted in Figure 5. The optimal balance of three points connected by equally weighted edges, in this case forces, is an equilateral triangle; as Figure 6 shows, the three vertices A, B, and C form a balanced equilateral triangle after the optimization is performed. The two nodes, D and E, that could be called the outliers of the graph because they only connect to one of the other nodes, are clearly spaced away from the other nodes. This will become even more evident on a larger data set.



Modularity = 0.00
Eps = 0.00

Figure 5: Example Graph with Five Vertices

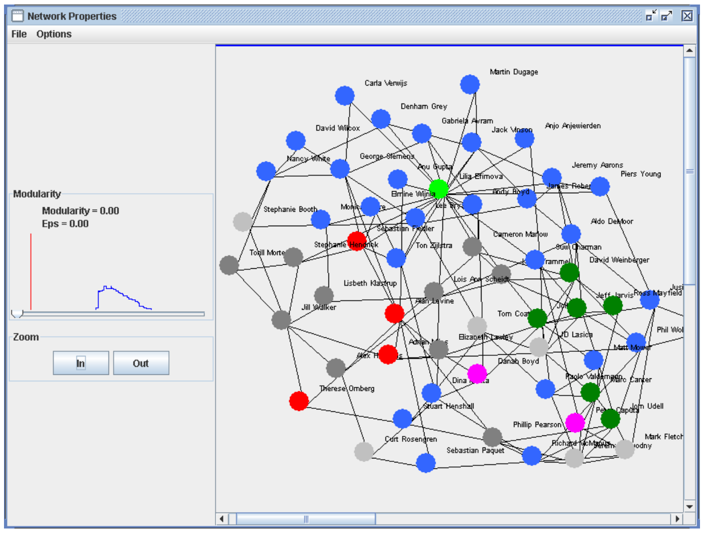


Modularity = 0.00
Eps = 0.00

Figure 6: Graph with Five Vertices after Optimization

## 5.2 Data Set Analysis

Once the repositioning aspect of the routine is understood, it is necessary to apply the algorithm to real data. The first data set is built from the relationships of online bloggers, people that keep an online journal on either a specific or a variety of topics. There are websites that offer to host blogs and although sometimes these are organized by topics, the source for this data set was hosted by an all-purpose blog site[1]. Each vertex in the graph represents a blogger, and two bloggers are connected with an edge if at least one has mentioned the other in the past month of blog entries. The vertices are colored depending on the website that hosts their blogs, and the expected result of running the grouping routine is that the vertices will be clustered by color. In practice, the resulting graph shown in Figure 7 does not provide the strong sense of which vertices are clustered together that had been anticipated. This seems like a disappointing result, yet analyzing the nature of the data exposes the source of the unexpected result. An assumption was made by expecting two bloggers to reference each other if their entries are hosted by the same website. This assumption represents a relationship that is not true in the data set, and so the results in Figure 7 are not a complete failure of the algorithm but rather a misunderstanding of the data. There are no trends to be found in the data set, as Figure 7 shows, and running the data through the tool allowed the viewer to comprehend a fact about the data that was not initially evident.



Modularity = 0.00
Eps = 0.00

Figure 7: Graph of Online Bloggers after Clustering Routine

An alternative data set to show the application of the visualization tool to social networks is shown in Figure 8. The graph is built from the friendships of a person named Tina [2], where the people are depicted as vertices and an edge connects two friends. The relationships in this graph are defined, not assumed as in the case of the data set built from bloggers, and the resulting graph after the optimization executes is clearly partitioned, as seen in Figure 9. In addition to the partitions, however, it is possible to quickly recognize the vertices that are not members of any cluster. In Figure 9 there is an obvious outlier in the node marked as Ana, and a hub labeled Julie that has an equal number of connections to both of the large subgraphs. This network is a prime example of the ability of this optimization approach to enhance a trend that would otherwise be unseen, as is clear in Figure 8. In fact, the trend displayed is not an easy one to identify by analyzing the data set element by element; the graph built on Tina's friends only has fourteen vertices, yet a graph built from a social networking source such as Facebook can easily have as many as sixty-nine million vertices [5].

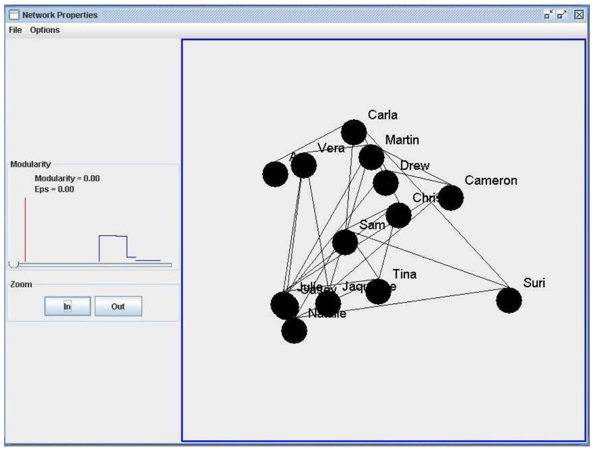


Figure 8: Graph of Tina's Friends

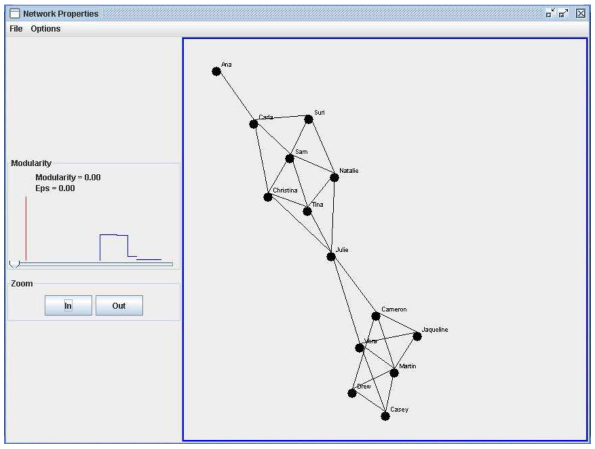


Figure 9: Clustered Graph of Tina's Friends

The final data set presented to demonstrate the ability of the visualization to spatially separate subgraphs was generated from the NCAA Football Bowl Subdivision, which used to be Division 1-A, 2006 schedule of matches. Each team is represented by a vertex in the graph, and two vertices are connected if there is a scheduled match between them. There are 180 teams, including teams belonging to lower divisions that were scheduled against a Division 1-A team, and 787 matches in the data set. The vertices are colored by conference, and the teams belonging to a conference outside of Division 1-A are colored a light grey. When graphed, the data is jumbled and even though the nodes are colored it is difficult to pick out a single conference as shown in Figure 10.

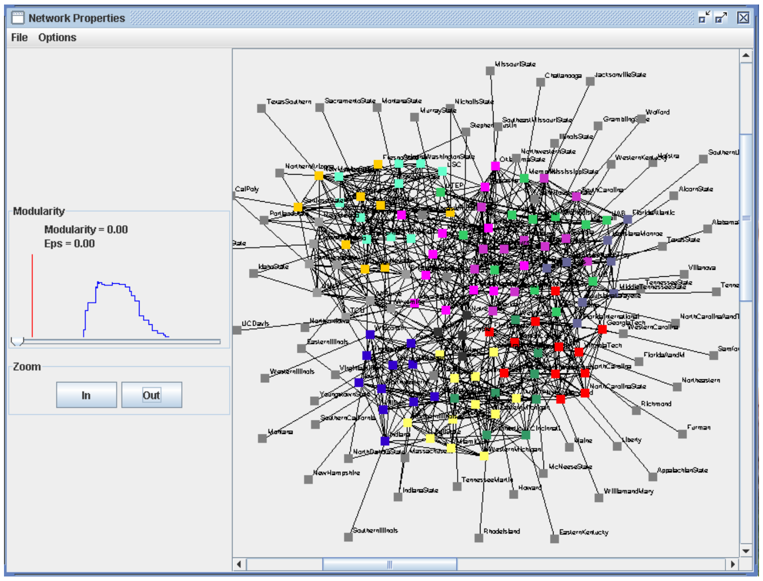


Figure 10: 2006 NCAA Division 1-A Football Schedule

The large size of this data set makes the threading part of the implementation evident, and as the computations take place the display is repeatedly repainted with the most recent graph. The viewer actually sees an animation of the computation, and can visually track the paths of vertices as they are repositioned. Figure 11 is a screenshot of a moment in the animation, and it can be appreciated that the animation contributes to the visualization by providing the audience a growing appreciation of the subgraphs as they become more and more obvious.

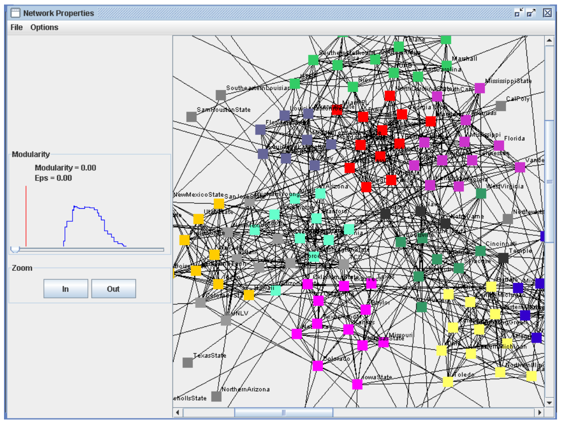


Figure 11: A Moment in the Animation of the 2006 NCAA Graph

The final outcome of the graph is shown in Figure 12, and the most striking vertices are the outliers and the hubs as they can be quickly visually spotted as anomalies or points of interest in the data.

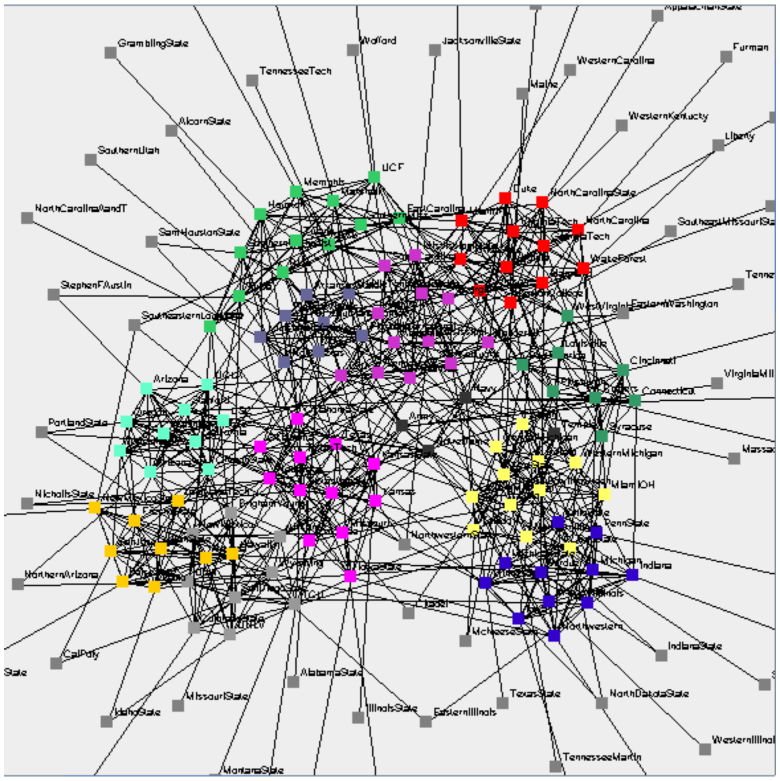


Figure 12: The Clustered Version of the 2006 NCAA Graph

# 6. CONCLUSIONS

## 6.1 Summary

Graph partitioning, or clustering, is useful in many fields of science and engineering to better understand patterns in the data that are not as clear by analyzing the elements individually. Data sets can also be considerable in size, as in the overwhelming task of identifying coexpressed genes in a set of 13,600 fruit fly genes [13]. Even the use of computing power does not assist if the clustering algorithms available cannot be applied to a specific type of data, as is demonstrated by ecological studies that attempt to evaluate the relevance of different computational grouping techniques to their science's data [3]. This thesis details the design and implementation of an approach to graph partitioning that can provide visual insight into an attribute of data samples that other clustering algorithms overlook. Data sets are not only made up of subsets, but also of elements that follow none of the trends or elements that exhibit characteristics of more than one cluster. Using sample data sets, one constructed from authors of online blogs, another from a small social group, and the third built from the NCAA College Football 2006 Schedule, the routine's ability to visually enhance the subgraphs when these exist was demonstrated. The implementation of the routine provides visual feedback to its user through the length of its computation time that is at all times highlighting the groups, outliers, and hubs in the graph. In this sense, the execution of this research successfully addresses the purpose of visualization, to provide insight into a phenomenon of the data that is not obvious, while at the same time ensuring that at all times the visual content contributes to the appreciation of an underlying trend. Research, implementation, and demonstrations have all contributed to meeting the goals of this thesis.

## 6.2 Contributions

The work in this thesis contributes to a research project on improving grouping techniques at Acxiom Corporation. The preexisting application that served as the setting for the clustering visualization benefited from the enhanced presentation of the data sets. The tool on the sidebar of the application that allowed the viewer to turn edges off that did not meet the selected similarity level (Section 3.2) is largely dependent of the display of the graph, and organizing the vertices into clusters provides a better visual understanding of the capability of the tool. The field of data clustering could also be positively affected by the research of this thesis as the approach to recognizing the groups and special vertices can be extended from this visual use to a routine that runs behind the scenes. Although this implementation is not ground breaking in its use of graphical display, the goals of visualization that were met make it a model for designing a product around the existing conditions, the target audience, and the scalabiltity of the problem.

## 6.3 Future Work

This thesis provides the basic structure for a visualization of a clustering routine, yet the implementation can be improved in some areas. One way to strengthen the visual effect of the grouping would be to color the vertices by clusters. This would allow the success of the clustering technique to also be seen through the resulting color sets, and the outliers and hubs would stand out if they were painted a different color. The implementation also needs to consider an exception for the vertices or clusters that are completely separated from the rest of the graph. The current routine will move disconnected parts of the graph to a remote position because there is no restoring force to counteract the repelling force between the vertices. However, it is not necessary to position these elements as distant as possible from the rest of the graph for the viewer to be able to see that they are disconnected. Additionally, the implementation sets limits in the initial repositioning of vertices and in identifying the equilibrium point, and both of these limits might affect the performance of the routine depending on the size of the data set. The vertices must be repositioned before any forces are calculated in order to avoid the possibility of two vertices being exactly in the same position. The current implementation sets horizontal and vertical boundaries to the area where the vertices will be positioned, yet if a data set is significantly large it could be benefitial to expand these boundaries so that the first iterations of force calculations do not result in extremely large repelling forces due to vertices located too close together. In the case of the equilibrium point, a larger data set has more forces that need to reach a balance, and reaching a point where the size of the largest move is smaller than the set limit might require more iterations than are truly necessary. Finally, the work that would be of most benefit to this application is to analyze different data sets across a larger variety of applications on the technology.

# REFERENCES

[1] Anjewierden, A. (2007, September 30). Retrieved February 2008 from <http://anjo.blogs.com/metis/gifs/aoir_text5.gif>

[2] Batagelj, V; Mrvar, A. (2007, February 24). Networks. *Program for large network analysis*. < http://vlado.fmf.uni-lj.si/pub/networks/data/>

[3] Belbii, L.; McDonald, C. (1993, April). Comparing Three Classification Strategies for Use in Ecology. *Journal of Vegetation Science,* Vol. 4, No. 3 pp 341-343.

[4] Delap, S. (2005). Swing Threading. *Desktop Java Live*. Ch. 5 pp 147-188.

[5] Facebook Press Room. Retrieved March 2008 from <http://www.facebook.com/press/info.php?statistics>

[6] Kovacs, J. (2001). Coulomb's Law. *Project PHYSNET*. <http://www.physnet.org/modules/pdfmodules/m114.pdf>

[7] Kurtus, R. (2005, August 16). Electrical charges. Retrieved July 2007 from <http://www.school-for-champions.com/science/electrical_charges.htm>

[8] Owen, G. S. (1999, February 11). Definitions and rationale for visualization. <http://www.siggraph.org/education/materials/HyperVis/visgoals/visgoal2.htm>

[9] Nave, R. Elasticity. *Periodic motion*. Retrieved on July 2007 from <http://hyperphysics.phy-astr.gsu.edu/hbase/permot2.html>

[10] Noack, A. (2007, February). Energy Models for Graph Clustering. *Journal of Graph Algorithms and Applications*. Vol 11, No. 2, pp. 453-480.

[11]Sanden, B. (2004, April). Coping with Java Threads [Electronic Version]. *IEEE Computer Society*. Volume 37, Issue4, pp 20-27. <http://ieeexplore.ieee.org/iel5/2/28844/01297297.pdf>

[12] SUN Microsystems. *JavaTM 2 Platform Standard Edition 5.0 API Specification*. Retrieved July 2007 from <http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Object.html>

[13] Tegner, J.; Bjorkegren, J. (2006, November 13). Perturbations to Uncover Gene Networks. *Science Direct*. <doi:10.1016/j.tig.2006.11.003>

[14] Wiswedel, B., Patterson, D., Berthold, M. (2003). Interactive Exploration of Fuzzy Clusters Using Neighborgrams. *The IEEE International Conference on Fuzzy Systems*. pp. 660-665.

[15] Xu, X., Yuruk, N., Schweiger, T. (2007, August 12). SCAN: A Structural Clustering Algorithm for Networks. *ACM*. pp. 824-833.

[16] Zhang, K. (2007, May). From Abstract Painting to Information Visualization. *Visualization Viewpoints*. pp. 12-16.