A NOVEL SCHEMATIC FOR THE REPRESENTATION AND ORGANIZATION OF ABSTRACTED DATA

Julia Ferraioli

An Undergraduate Thesis

in

Computer Science

Presented to the faculties of Bryn Mawr College in partial fulfillment for the Senior Thesis requirement toward a major in Computer Science.

2007

ABSTRACT

A Novel Schematic for the Representation and Organization of Abstracted Data

Julia Ferraioli

This paper proposes an improved solution to the problem of an ordinary user navigating through and organizing large data sets of possibly high dimensionality. The schematic described within takes a twofold approach. The first aspect involves the representation of the data set to the user. We assume that the user has neither the time nor the inclination to examine every piece of data individually, and therefore an abstracted representation of the data set would be appropriate. We explore how to make a "smart" visual representation of each piece of data, keeping in mind that the user has some background knowledge of the information being represented.

The second aspect involves how to help the user arrive at a correct organization of the data in a shorter amount of time than it would take to classify each data item manually. We combine and modify several existing machine learning techniques and apply them to the problem of data classification when the user has background knowledge that can be applied to complete the task. This approach may be used when the user has a specific goal in mind as well as when the user only has a vague idea of how he or she would like the data to be organized. In the latter case, we can look upon the clustering algorithm as a suggestion tool instead of a tool to produce the desired result more quickly.

By combining an intuitive graphical user interface with a classification tool that incorporates user feedback, we believe that it aids the user in arriving at the desired goal more quickly than any existing methods. The composite schematic is what we describe in the paper as well as the results of experiments testing the effectiveness of the proposed design on data set of varying degrees of difficulty.

Contents

List of Figures

Chapter 1

Introduction to Problem

1.1 Introduction

Large data sets, by their inherent qualities, are difficult to interpret. This is especially true when the data set is composed of datum with which multiple qualities are associated. When the number of qualities, or attributes, becomes high enough, we term such a data set as "of high dimensionality." For example, we use a data set which characterizes wine through 13 characteristics such as alcohol content, magnesium content, flavanoids and more. Data sets with higher dimensionality are even harder for the human to analyze without aid, due to the fact that they have more attributes to take into account. Large data sets with high dimensionality are one of the reasons that there are so many tools available to analyze and interpret them with no additional for user input.

However, when the user does have background knowledge necessary or desirable for the correct analysis of that data set, we should take into account their desire and need to make appropriate use of that knowledge. One way is to let the user manually decipher the data set, but that approach is time consuming and wasteful of resources. What is needed is a way to let the user maneuver with or without purpose throughout the data set, and have the system interpret the actions of the user in order to arrive at the correct analysis with the least amount of effort. This way, the user is able to contribute to the system in order to see their desired end result.

During this paper, we will explore a way to allow the user to incorporate this knowledge into the arrangement of data sets using a graphic user interface (GUI) combined with incremental algorithms. We will present the schematic, the data sets used for analysis, the experimental designs, and the results. The results will support our hypothesis that the combination of a GUI and user contribution with machine

learning enables the correct arrangement of a data set more quickly than would otherwise be possible.

1.2 Motivation

The goal of this research is to develop a way to allow users to organize and partition data into clusters that will aid in achieving their particular goals. Traditional clustering algorithms look solely at the characteristics of each piece of data, attempting to group the most similar items together. However, the end result from such a clustering algorithm may or may not fit the user's needs. The user may wish to minimize the contribution of certain characteristics and emphasize other characteristics from a data set. Most existing clustering algorithms are not able to do this without the help of other algorithms such as dimensionality reduction algorithms, and even then they are not able to take input from the user.

While several clustering algorithms exist that are able to incorporate user feedback [BBM04], [WCRS01], they all rely on a predetermined value for the number of desired clusters. While this may be acceptable if the data set in use is one which has already been analyzed beforehand or if the user has a specific goal in mind, most data that needs analysis has not been closely examined. Therefore, knowing the correct number clusters is an unreasonable expectation. We hope to create a machine learning algorithm that is not dependent on knowing the number of clusters beforehand, and that also takes feedback from the user. To the best of our knowledge, there is no existing schematic that incorporates user feedback with a growing algorithm.

Similarly, there has been very little research done in the area of smart data representation, probably for the reason that there is not much that one can do with only the representation of data. There is also the problem of how to actually represent the data. When the dimensionality of the data set is low, say two or three characteristics, it is easy to see how we might depict such data. Perhaps we assign each attribute to a red green or blue value, thereby yielding unique but meaningful color values for each piece of data. This problem becomes much more difficult when we see data sets with high dimensionality; then the problem is encountered of how to pick and choose characteristics to use when deciding the representation of the data. However, if the representation of the data is abstract, yet meaningful, we believe that it will be easier and more pleasant for the user to interact with this representation.

The two problems of organization and representation of data may be independently researched. The absence of one does not make the problem of the other any less valid or challenging. The new learning algorithm has no requirements about how the data is represented. It does require the user to be able to interact with the data, which is where representation comes into the picture. While the two problems may

be independent of each other, we believe that including a smart visualization will aid the clustering algorithm in that a better representation will allow the user to select nodes that give the most useful constraints. We believe that combining the solutions to the two problems will result in an intuitive interface that is both effective and easy to use.

Chapter 2

Related Work

In this chapter, we will explore some of the fields which have an impact on our topic. The two main areas we will be describing are sub-fields of machine learning and algorithms for dimensionality reduction. We will then briefly cover what work has been done in the area of incorporating user feedback and machine learning with a visual interface.

2.1 Machine Learning

Machine learning is a sub-field of artificial intelligence that attempts to allow machines to "learn" how to solve problems. Different machine learning algorithms target different types of problems. For example, neural networks attack problems that may not be conducive to linear solutions or where the task is simply too complex to be done by hand, whereas clustering algorithms take tasks that may be performed manually, but is often too tedious and time consuming to do so. Most machine learning algorithms are involved in the task of information generalization, which follows what humans do when attempting to learn a task. Whether the goal is to find patterns in information, or to make a complicated task easier, generalization comprises most of the aim of the algorithms we will be reviewing.

2.1.1 Generalization Algorithms

Because we are dealing with partly a classification problem, the algorithms emphasized below are generalization algorithms. Primarily two different types of clustering algorithms exist: hierarchical clustering and partitional clustering. Due to the nature of the graphical interface, we consider only partitional clustering algorithms. Parti-

tional algorithms take a data set and partition it into categories (clusters) based on a set of criteria.

k-Means

The most well known clustering algorithm is MacQueen's k-means [Mac67]. It is often held as baseline or benchmark when comparing the effectiveness of clustering algorithms. Much of its appeal lies in its simplicity; the algorithm is not only easy to understand but also easy to implement. K-means partitions a set of n pieces of data into k clusters based on each datum d_i's feature vector, with $0 \leq i < n$. A feature vector is the representation of the datum's attributes. Each datum's feature vector is interpreted as the position of the datum in the attribute space, where the attribute space is the n-dimensional space where we place all feature vectors.

K-means starts out with k cluster centers randomly positioned in the feature space. Each datum d_i is assigned to the closest cluster center, based on some distance measure. The cluster centers' feature vectors are then updated to be the average of the feature vectors of the data items assigned to them. By updating the cluster centers' feature vectors, we essentially move the cluster centers to be in the middle of the data items assigned to them. Data items are reassigned and cluster centers are updated until convergence or some maximum iteration variable is reached.

Although it is a very intuitive algorithm, k-means does have its intrinsic disadvantages. Because initial cluster centers are chosen completely randomly, it therefore may take much longer to converge to the correct clustering than if they were chosen using a different method. Also, like many clustering algorithms, k-means is dependent on the value of k. The clustering algorithm assumes a correct value of k in order to work optimally. An incorrect k would cripple the classifier into getting entire classes of data incorrectly positioned; it would group data into clusters in which there is only nominal similarity, which would give a false sense of membership.

Neural Gas

Neural Gas [TMMS93] is a relatively new algorithm so named because the feature vectors' movement resembles the movement of gas particles. This algorithm involves k feature vectors (cluster centers), much like k-means, which traverse the data set, learning as they move. It is an on-line algorithm, which means that the feature vectors are updated before the whole data set is presented.

A data item d_i is randomly presented to the feature vectors, and the feature vectors are sorted by their distance from the currently selected data item d_i. The closest feature vector and its neighbors are adjusted based on d_i's feature vector and

a learning rate, a neighborhood function and a decay factor. Then, each data item is assigned to it's closest feature vector.

Neural Gas suffers from some of the same weaknesses as k-means. It is entirely dependent on a correct value of k as it cannot create new feature vectors or delete unused ones. In addition, while the fact that neural gas is an on-line algorithm adds to the overall accuracy of the algorithm, it also adds to the complexity of actually computing it. It may become unsuitable when attempting to visualize its results on-line as well, depending on the complexity and overhead of the visualization.

2.1.2 Incremental Learning

Incremental learning allows the user to specify a beginning criterion but leaves it up to the learning system to add ways of increasing functionality if so needed. In a clustering context, incremental learning lets the algorithm add cluster centers if the algorithm determines that accuracy would improve with an added feature vector. This type of approach is effective for eliminating the problem of choosing the correct k for an unknown data set, though it adds complexity to the overall algorithm.

Growing *k*-Means

As a variation on k-means, Daszykowski et al. introduced their algorithm Growing k-Means [DWM02] which takes advantage of the incremental learning approach. Growing k-means essentially turns k-means into an on-line algorithm. The user still must specify the maximum number of cluster centers possible, but the algorithm starts with only two.

The starting centers are randomly positioned in the feature space, and a random data item d_n is presented to the centers. The closest center is determined, and that center is moved fractionally to d_i. Once the whole data set has been seen by the cluster centers, each data item is assigned to a center. The distance between each data item and its assigned center determines where the new cluster center will be positioned.

The same authors also introduced Growing Neural Gas which turns neural gas into a dynamic on-line algorithm. The algorithm remains mostly the same, except for the introduction of a new feature vector (cluster center) and the introduction of edges into the framework.

2.1.3 Incorporation of User Feedback

Some research has been done on how to incorporate user feedback into clustering algorithms. It is a field that is only now starting to get attention as new ways to interact with a computer are invented. Most of the existing research has been on ways to modify k-means to include a set of constraints. Such algorithms are called semi-supervised clustering algorithms, due to the fact that the user gives them sporadic (as opposed to constant) feedback. We will review two such algorithms.

COP-Kmeans

COP-Kmeans was introduced by Wagstaff et al. [WCRS01] as a way to incorporate user defined constraints into k-means. The algorithm enables the user to define two types of constraints: must-link constraints and cannot-link constraints. This allows the user to tell the algorithm that certain pairs of data items absolutely should or should not be assigned to the same cluster center.

The COP-Kmeans algorithm simply adds an extra set and step to the traditional k-means algorithm. Before assigning a point to a cluster center, it ensures that doing so will not violate the set of user defined constraints on that point. By adding this step, we are left with a set of clusters that follows the set of user constraints completely, or else the algorithm fails altogether.

This algorithm introduces an effective way to take into consideration user preference when performing clustering. However, besides the traditional k weakness, if the data set is over-constrained by the user, the algorithm fails entirely. If the user does not make any mistake, then this should not be a problem, but if the user has a misunderstanding about the data in question, then this might result in an over-constrained problem and thereby a failed clustering.

PCK-Means

Pairwise Constrained K-Means is an algorithm introduced by Basu et al. [BBM04] a few years after the introduction of COP-Kmeans. PCK-Means also uses a set of must-link and cannot-link constraints, defined by the user. However, unlike COP-Kmeans, this algorithm is allowed to violate the constraints.

Each violation is assigned a penalty, and part of the goal of the algorithm is to minimize that penalty. If no violation is necessary, then PCK-Means will find the clustering that does not violate the constraints at all. However, in the case of an over-constrained data set, the algorithm will not fail, but make the violation that

costs the least. Other than the penalty-minimization, PCK-Means performs typical k-means clustering.

PCK-Means clustering has a distinct advantage when the user might be unsure about the characteristics of the data set in question, because it cannot fail due to over-constraint. However, the introduction of the cost function adds complexity to the algorithm which could be detrimental in various situations. Overall, the algorithm allows the user to give feedback on a clustering, without assuming perfection on the part of the user.

2.2 Dimensionality Reduction

Dimensionality reduction is an extensive area of research for information retrieval experts. It involves the problem of how to determine which dimensions contain the most important information in a feature vector. For instance, is the zero-filled dimension more important than the one with varying values? Dimensionality reduction tries to determine which has more bearing on the data as a whole. When done properly, dimensionality reduction results in a compact description of a data set without unrelated or unimportant variables.

2.2.1 Principal Component Analysis and Singular Value Decomposition

Principal component analysis (PCA) and singular value decomposition (SVD) are both statistical methods for reducing data of high dimensionality down to low dimensionality. When all feature vectors have been combined into one large matrix, a series of transformations performed on that matrix results in a smaller, more manageable matrix. Though the transformations performed by PCA and SVD are different, they yield similar results.

Applications for PCA and SVD range anywhere from gene analysis [MRJ87] to computer vision to robotics. In machine learning it is especially useful in determining so-called "noisy data" as well as determining the important dimensions in clustering [KL97]. For our purposes, we are interested in PCA and SVD primarily for their uses in determining how to visually depict data items.

2.3 Visual Incorporation

Some research has been done in the area of visualizing information on screen in order to cluster it. For the most part, they have dealt with k dependent algorithms, which is fragile if k is not correct. However, they comprise the building blocks of the research presented in this paper.

2.3.1 Interactive Partitioning

The system demonstration of Interactive Partitioning by Lesh et al. [LMP01] shows their approach to combining user feedback with a visual interface. They attempt to solve the NP-hard problem of k-way network partitioning. While this schematic deals with a very specific type of problem, their approach to solve it resembles our own approach to solving a more generalized type of problem.

They visualize the hypergraph of the network, and let the user edit the partitioning on screen. Afterwards, the system is able to refine the network based on the modifications performed by the user. The visualization is done using a force-directed approach with spring embedding, which determines the position of each node.

2.3.2 Interactive Visual Clustering

Interactive Visual Clustering (IVC) [dFM07] is the schematic which comprises most of the basis for this paper's research. In this schematic, the clustering algorithm used is Basu et al.'s PCK-Means, along with relational clustering. Constraints are generated by interpreting the user's movements of data on-screen, clustered and then the graphical representation is updated.

Preliminary results are promising that this schematic results in a correct clustering faster than traditional clustering algorithms and visualizations. User studies are pending funding and approval. The primary contribution of this paper was the incorporation of a visual interface with a clustering algorithm that was supplied with constraints incrementally. Details not supplied here will be included later.

Chapter 3

Program Background

The program's schematic has many parts that are integral to the success of the fusion of visual representation and the behind-the-scenes algorithm. In this chapter, we will outline in detail all the components of the program that contribute to the overall schematic. We will first concentrate on the data format and conversion, and then go into the details of how the visualization works, and lastly show step by step what a typical interaction might involve.

3.1 Terminology

There are a few terms that should be clarified before progressing. When referring to a data item, we will call that a "node" in the overall graph. Each node has an associated feature vector, which is the list of n attributes in vector form. Two types of spaces exist in our schematic, and the two may be easily confused. First, there is the attribute space, which is the n-dimensional space associated with the nodes' feature vectors. Secondly, there is screen-space, which is simply the two-dimensional space we see on the display. The two spaces are independent of each other.

From here on, we will use the n to index into each node's feature vector. We will use i to refer to each node (d_i) individually. The letter k will continue to indicate the number of clusters. Other subscripts will be defined as needed.

```
<node id="0">
<data key="$val0">1</data>
<data key="#val1">5.1</data>
<data key="#val2">3.5</data>
<data key="#val3">1.4</data>
<data key="#val4">0.2</data>
<data key="ID">0</data>
</node>
```

Figure 3.1: An example of a node in GraphML format

3.2 Data Format

Although a standard data format is not necessary for the program, we believe that having one simplifies the process later on. It standardizes the type of parser used for reading in the data, so that the program does not have to determine the format beforehand and account for it. Also, a standardized data format is useful in determining the nature of the data at first glance.

3.2.1 GraphML

The data format that we have chosen to use is the GraphML format (Graph Markup Language) which is based on the extensible markup language (XML). We feel that GraphML is a natural choice because it can easily represent a graph with any number and type of attributes per node, as illustrated in 3.1. The graph format in which it represents the data easily transfers to what is seen on screen. GraphML was initiated by the Graph Drawing Steering Committee as a natural way to represent graphs. As graphs (a collection of nodes and possible edges) are the only type of data we are considering, GraphML seemed like the natural choice for a data format.

Conversion Tool

However, GraphML is not widely in use. The most common data format is what is known as comma separated value (CSV) data. To get around that, we created a little program that converts CSV data sets into GraphML specification. This actually has an added benefit, which is that with this program, we can standardize all attribute names, so that they are not data set dependent. Standardized attribute names aids in making the program data set independent. The first attribute value is always the value which identifies to which cluster the item should belong, so we preface that attribute label with a "$" to indicate that it should be ignored for all purposes except

when evaluating cluster assignments. It is defined as the first attribute in order to make the conversion easier; if the user had an already defined data set in GraphML format, this would not be necessary as it would just use the attribute label prefaced with a "$" as the class label. All other attribute labels we preface with a "#" to indicate that they are values to take into consideration for clustering.

3.3 Visualization Specifications

In this section, we will describe the visualization specifications that allow the user to be able to interact with the program effectively. This includes what handles the visualization and the specifications of factors in the visualization. Most of the specifications such as constraint generation and the use of force-directed layout builds on the design of IVC [dFM07]. We will limit our explanations to concepts and not the physics behind the concepts.

3.3.1 Prefuse

To create our visualization, we decided to use the interactive information visualization toolkit Prefuse [HCL05]. Prefuse is a package written in Java that allows for many different types of visualizations, including different layouts, node representation, and visual effects. Although our program is written as an off-line application, it would be simple to extend it to applet format so it may be placed on the Internet.

3.3.2 Force-Directed Layout

A force-directed layout is chosen as our underlying layout. This type of layout uses the principles of physics in order to create an aesthetically pleasing visualization. Each node is modeled on an electrically charged particle with an associated repulsion or attraction force. An edge between two nodes is modeled after a spring. When using Prefuse, one can set the nodes to be as repellent or attractive as needed, which may be useful for different purposes. Edge tension is also fully customizable; one can make it so that edges have no tension at all, or they can attempt to have no length at all.

In Figure 3.2, we see a sample graph generated using Prefuse's force-directed layout with a social networking data set. While using a layout such as this one, the graph is constantly attempting to settle in the "optimal" position. It attempts to position the nodes such that no nodes are repelling or repelled, and such that each spring is at its desired length. Depending on the amount of data involved in the

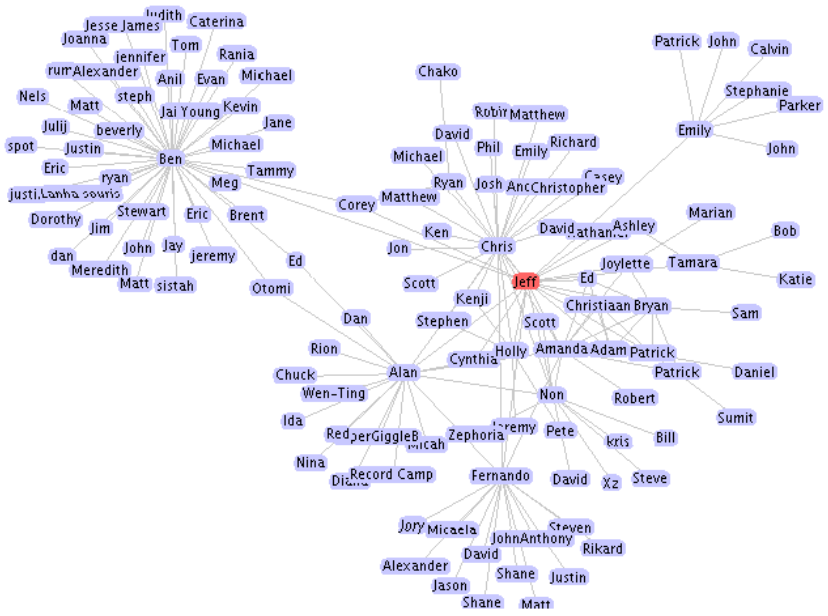

Figure 3.2: An example of a graph drawn using force-directed layout

graph, this may or may not be possible. If it is possible, then the visualization will keep iterating until that equilibrium is reached. If it is not possible, then the user will see the visualization keep adjusting; never settling.

3.4 User Interaction

The user has multiple ways to interact with the interface. She is able to click on nodes and position them around screen. If edges exist, nodes connected immediately and peripherally with the selected node will also be repositioned. Besides direct interaction with the nodes, the user is able to move the entire graph around by panning, or zoom in or out on the graph. Panning and zooming are very useful when the data set is large and cannot be displayed on-screen at one time.

3.4.1 Constraint Generation

We concentrate mostly on the way that the user interacts with the individual nodes. While panning and zooming are nice features, they are not integral to this proof-of-concept. Many ways of interpreting user actions have been proposed, such as click

frequency, duration and other methods. We feel that the only information that can be truly inferred from user action in this context is the similarity of nodes.

The way we see it, when the user moves a node around on-screen, she is communicating information about that node in relation to other nodes which she has already moved. Intuitively, we determine that if the user moves a node far from others already placed, she says that the node most recently moved is dissimilar to the other nodes. If she places it close to other nodes already placed, then she communicates that those nodes are similar.

These are the factors by which we generate constraints. Specifically, if the distance between node d_i and d_j, where j is the index of the fixed node to which d_i is being compared, is less than δ, then a "must-link" constraint is generated. If the distance is greater than ϵ, then a "cannot-link" constraint is generated. If the distance is greater than δ but less than ϵ, no constraint is generated. This allows there to be neutral territory on the screen if the user is not quite sure where a node should go. The constraints are stored in an $n \times n$ matrix for quick recall.

3.5 Sequence of Events

In this section, we will describe the sequence of a typical interaction with our visualization. This is to illustrate the steps involved in creating the on-screen display, so we will not be detailing the intricacies of algorithms working behind the scenes yet. Details for the clustering algorithm will follow in the subsequent chapter.

3.5.1 Initial Visualization

The initial visualization positions the nodes on the screen. Contrary to what may seem logical, the reasoning for the initial placement of the nodes has nothing to do with the nodes' feature vectors. Instead, it simply relies on the physics behind the force-directed layout and the spring embedding to get the nodes to an equilibrium, as illustrated in Figure 3.3. As a result of the forces involved, if there are no constraints on the nodes, such as edges between nodes included in the data set, nodes have a tendency to drift to the edges in the initial stages.

3.5.2 First Node Moved

User interaction begins when the user moves one node. Intuition would indicate that the user would move a node to the edge of the screen, but this is not a requirement.

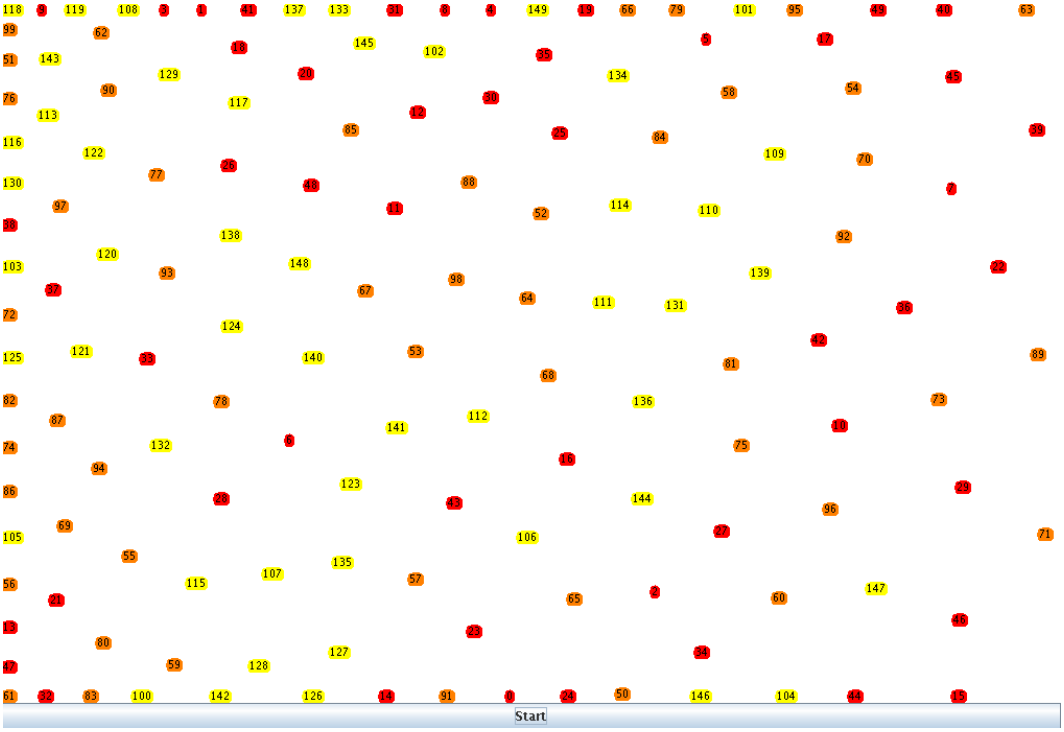

Figure 3.3: An initial visualization of the iris data set

Once a node is clicked, we immediately stop the physics surrounding the visualization because the user should be able to move a node in the same context in which she selected it. After the user releases the node to rest in its final position, the screen coordinates of that node are recorded in a matrix. Once the node has been released, it stays fixed in that position and we restart the physics engine. This has the effect that the repulsion force of other nodes cannot move it, but rather it repels all other nodes which come near it. We also increment the count of nodes moved, as this is important for generated constraints later.

3.5.3 Second Node Moved

When the user moves the second node, we increment the count of nodes moved again, and record the final coordinates of this node. However, now that we have two sets of coordinates, a constraint can be generated. We look at the node just moved, and calculate the distance between it and the first node moved. Depending on the distance value and the values of δ and ϵ, we generate the appropriate constraint and insert it in the constraint matrix. Typically, the values of δ and ϵ are generated based on the size of the screen, though other methods of choosing their values may be used as well.

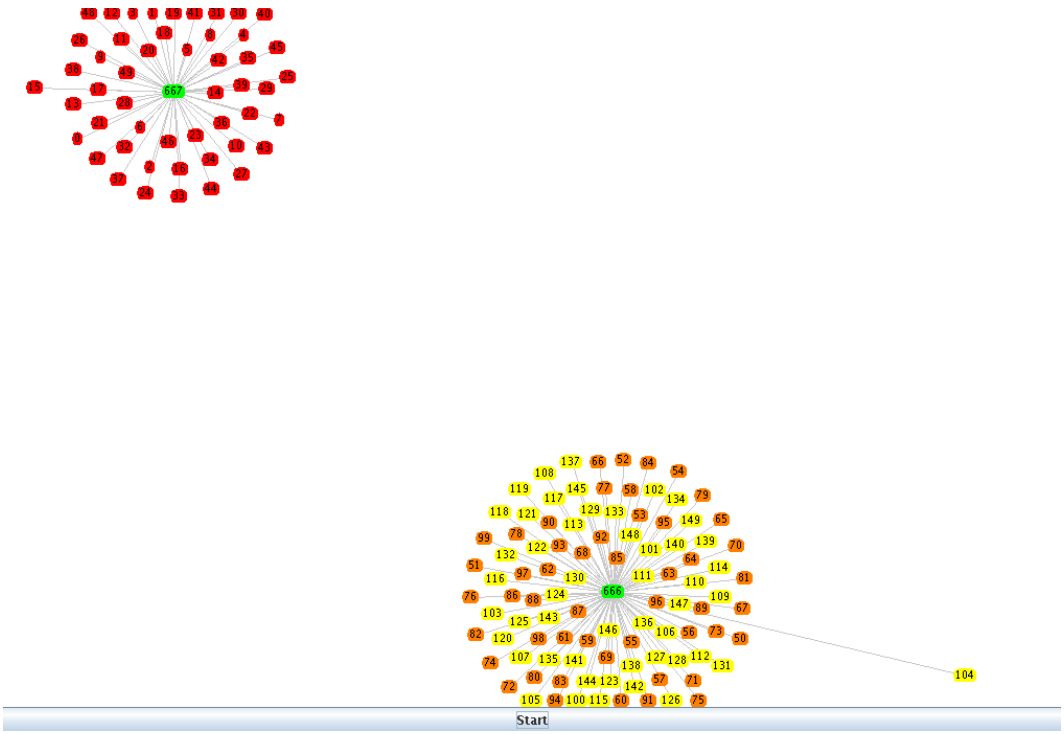

Figure 3.4: A visualization of the iris data set after initial clustering

3.5.4 Calling of Clustering Algorithm

Now that we have an entry in our constraint matrix, there is enough information to run the new constrained clustering algorithm. When calling the clustering algorithm for the first time, we cluster based on the constraint matrix, the feature vectors for all of the nodes and the current value of k. At this point, k is set to 2 and is later incremented. The details of our clustering algorithm will be explained later. Once the clustering algorithm is finished, we are left with a list of cluster centers and also know which nodes belong to which clusters.

3.5.5 Creation of Cluster Centers

To reflect the clusters on screen, we needed a way to bring the clusters together visually. In order to do this, we create dummy nodes to represent the cluster centers. Each center has its own unique identification number and class label in order to give it a unique coloring. Iterating through the nodes, we attach each node to its assigned cluster center with an edge. In case there are any edges in the original data set, we assign these "cluster" edges to have a heavier weight and a shorter default optimal length so that they will have more effect on the graph than regular edges.

As we can see in Figure 3.4, the two fixed nodes draw the cluster centers towards themselves. In this example, the first two nodes drawn have a "cannot-link" constraint between them, so there are two distinct fixed clusters. The cluster centers are represented by the bright green nodes, and are being drawn to the fixed nodes; one which is yellow and one which is orange. As the cluster centers are drawn, so are the unfixed nodes attached to the cluster centers.

3.5.6 Subsequent Modifications

Any subsequent nodes moved has much the same effect as moving the second node. Before any of the typical steps are performed, we first clear out the storage for the cluster centers, as well as remove the cluster centers from the visual graph. In each case, additional constraints are generated, except that now the most recently moved node is compared against every previously moved node. This constraint matrix is now used in the clustering algorithm. After the clustering algorithm is finished, the dummy nodes are created, along with the edges to its members. The cycle of moving nodes and generating constraints can continue indefinitely, though it is logical that the user would stop after all the nodes are moved to some satisfaction.

Chapter 4

Clustering Algorithm

This chapter's goals involve explaining what motivated the development of a new algorithm, as well as describing the specifics of the algorithm itself. We will explore the main influences on our combination clustering algorithm and detail the assumptions made when deciding on certain aspects of the algorithm.

4.1 Goal for Clustering Algorithm

What prompted the development for a different clustering algorithm was the continuous dependence of various clustering algorithms on k. We had seen examples of where k was poorly chosen, thus skewing all resulting clusters. While various algorithms have been created to choose a proper value of k, these algorithms require a pre-screening of the data to find the number of "natural" clusters.

When the growing k-means algorithm was developed [DWM02], it seemed like a good step forward. However, for our purposes in combining it with a graphical user interface, it would not suffice. Growing k-means worked as an on-line algorithm, presenting the data set to the feature vectors incrementally, and we needed a batch algorithm. Also, a way was needed to take into consideration the user's background knowledge, and growing k-means had no built-in method to do that. These two considerations drove the development of our new clustering algorithm.

4.1.1 On-line versus Batch

There are two ways that machine learning algorithms can learn a data set: on-line and batch. On-line learning algorithms make modifications to the learned task before

seeing the entire data set, which may be seen as closer to what humans do. Batch learning algorithms only modify the learned task after seeing the entire data set. In this way, no part of the data set is "forgotten" after seeing other parts of the data set; it treats each piece of data with equal weight.

While we think that on-line learning is an effective tool, it is intrinsically unsuitable for our task for several reasons. First of all, as on-line learning is continuous, there is no real way to create cluster centers whose changes can be reflected visually. The reason for this is because too many steps go into reflecting those changes on the screen for it to be done many times per second. The system has to create the node, create all the edges and also remove the node once a new one is generated. Secondly, if we were to create it as an on-line algorithm, it would be impossible to incorporate user constraints, as doing so would require considering the entire data set. Thirdly, we believe that such an algorithm would simply be too computationally expensive for intensive visualization. Due to these considerations, we decided to make our new algorithm a batch algorithm.

4.1.2 Incorporation of User Constraints

For the incorporation of user constraints, we decided to choose between the method used by Basu et al. [BBM04] and the method developed by Wagstaff et al. [WCRS01]. Both use the type of "must-link" and "cannot-link" constraints that are generated in our program, and both have good experimental results. After careful consideration, we chose the COP-Kmeans constraining method.

4.2 Batch Growing COP-Kmeans

Our new algorithm, batch growing COP-Kmeans, is what resulted from this incorporation of user constraints with an incremental algorithm. We first detail the assumptions made about the data being analyzed and what is expected on the part of the user. Finally, we describe the resulting algorithm step by step.

4.2.1 Considerations

There are a number of assumptions that we made when designing this algorithm. These assumptions are in no way indicative of the way it must be; they are simply to make this proof-of-concept algorithm initially easier to implement. By making a number of assumptions, we are able to simplify the explanations and yet also describe how the algorithm can be extended.

Type of Data

For the purposes of a proof-of-concept, the type of data we allowed were of continuous value. With this criteria, we did not use categorical data. This was primarily to simplify the data structure and the distance function. If we had allowed for categorical data, it would have been necessary to incorporate a different distance metric, such as hamming distance. Although we excluded categorical data, it should not be taken as categorical data would invalidate the algorithm. Another distance function and different data structures could be implemented such that categorical data would be taken into account. It would not diminish the effectiveness of our batch growing COP-Kmeans, which we know because the IVC schematic [dFM07] incorporated categorical data.

User Competency

Another assumption we make is that the user is competent. For instance, we express confidence that the user is actually knowledgeable about the data set, because it would not make much sense for the user to be using this schematic if they are not. One reason we felt comfortable using the COP-Kmeans method for constrained clustering is because we assume that the user will not over-constrain the problem, which is the main weakness of COP-Kmeans.

4.2.2 Distance Metric

For the distance metric, we use Euclidean distance, otherwise known as the L-2 norm. Euclidean distance between x_i and $\mathrm{x'}_i$ is given by the formula:

$$\sqrt[2]{\Sigma(x_i - x'_i)^2}$$

In words, each attribute of the data item is taken as a dimension. We then find the distance based in n-dimensionality by subtracting one attribute from the other, squaring that and summing each squared difference. Afterward, we take the square root of that to find the overall distance between two data items.

4.2.3 Algorithm

We describe the details of our batch growing COP-Kmeans below.

1. Specify the maximum number of clusters, $maxK$.
2. Create two n-dimensional cluster centers c_0 and c_1 randomly positioned in the nodes' attribute space.
3. Present entire data set $D = \{d_0,\ldots,d_{n-1}\}$ to the cluster centers and assign each d_i to the closest cluster center such that VIOLATE-CONSTRAINTS is false.
4. Update all cluster centers to be the mean of the feature vectors of the points assigned to them.
5. Repeat steps (2) and (3) until convergence or a maximum number of iterations is reached.
6. If any of the nodes for each cluster center fall out of a distance threshold t and $k \leq maxK$, create a new cluster center, increment k and assign the outlying points to it.
7. Repeat step (3) and (4).
8. If any cluster center has no nodes assigned to it, decrement k and set $maxK$ equal to $k+1$.
9. Return the cluster centers $\{c_0,\ldots,c_k\}$.

The violate-constraints method is taken directly from Wagstaff et al. [WCRS01].

VIOLATE-CONSTRAINTS(node d, cluster c, must-link constraints $Con_= \subseteq n \times n$, cannot-link constraints $Con_\neq \subseteq n \times n$)

1. For each $(d,\ d_=) \in Con_=$: If $d_= \notin c$, return true
2. For each $(d,\ d_\neq) \in Con_\neq$: If $d_\neq \in c$, return true
3. Otherwise, return false

Explanation

The rationale for this algorithm is fairly straightforward, but we will review some of the trickier portions. Though this is a *growing* algorithm, we still need to specify a maximum k in order to bound in the final k. One could theoretically set the value of the maximum k to some absurdly large number to nullify the upper bound, but the variable would still be needed to settle on a final k, so that k and $maxK$ would be

equal to each other, indicating the correct value. This, however, is not a dependency on k, as k can still grow and shrink.

Step (1) through (5) are standard COP-Kmeans clustering, classifying based on feature vectors. Step (6) is where we create extra nodes based on whether or not there are nodes that seem like they belong *less* to their assigned cluster center than the other nodes. This belonging less to a cluster we define as when a node falls out of a certain threshold range from its cluster center. Once we find all such nodes, the algorithm assigns them to a new node which is located at the mean of all the outliers.

In step (8) we provide a way for the algorithm to detect whether or not it has exceeded the needed number of clusters. When a cluster center is assigned no nodes, we infer that it is not a useful center, delete it and reduce k by the number of unused centers. However, to keep the algorithm from making another such error, we also set $maxK$ to k+1. This lets the algorithm settle on a k that is possibly correct.

Chapter 5

Experiments

Experiments in the field of clustering are extremely difficult to conduct for several reasons. First of all, experiments are customarily judged by some "correctness" metric. With clustering, and especially constrained clustering, there may not be a "correct" clustering. The correct clusters to the user may in fact not be statistically correct. Therefore, when conducting experiments to judge the efficacy of clustering algorithms, we must take any results with a grain of salt, for it is always possible that subjective criteria may be present.

Secondly, we have to consider what makes a cluster more or less correct than another cluster. This may be entirely dependent on the user's viewpoint, and may vary from user to user. The degree of correctness is subjective as well. However, even though what makes a clustering correct may be completely subjective depending upon user and purpose, we still need a way to judge it. So while the experiments performed may seem somewhat artificial and contrived, we still conduct them in order to show that this is a viable algorithm.

5.1 Data Sets

Standard machine learning repositories exist for the sole purpose of judging how well a classification algorithm works. The one from which we have gathered our data is the UCI Machine Learning Repository [DNM98]. There are three data sets that we have decided to use for evaluation purposes. One is the standard iris data set, which has 150 instances, three classes and four attributes. Each class label refers to a type of iris, and the instances are distributed equally amongst the three classes. While this data set is known as a simple classification problem, it should be noted that only one class is linearly separable from the others due to the attribute values.

The second data set we selected is a wine data set. There are three classes, which identify three different types of wines grown in one region of Italy from their chemical makeup. There are 178 different combinations of 13 attributes. Although a classification technique has achieved 100 percent accuracy, that was a biologically-specific algorithm, not a generally applicable algorithm.

The third data set is a glass identification database. In this case, there are six different class labels, each corresponding to a different type of glass. The glass data set has 214 instances, each with nine attributes. These characteristics are not linearly separable. This is known as a difficult data set to classify. All three data sets have continuous attribute values, and each data item has a label so that we may compare our clustering with the "correct" clustering given by the class label.

5.2 Experiment Setup

The experiments that we use are not the typical experiments where we would compare against other types of clustering algorithms, for there are no comparable algorithms. Therefore, there are only a few tests by which we can judge the effectiveness of the algorithm. One factor by which would be desirable to judge the algorithm is the end number of cluster centers generated by our algorithm. If the number of clusters equals the number of class labels for the data, then we would consider that a success. Another factor is whether or not the algorithm classified the nodes correctly.

We designed four experiments to judge these two factors. The first experiment is to judge the effectiveness of the COP-Kmeans clustering algorithm when incorporated with our visual interface. We randomly select a node that has not already been picked, and move it towards its correct cluster. After two or more nodes have been moved, we run plain COP-Kmeans with k set to the correct value. The second experiment has the same setup as the second, except for instead of choosing a random node, we select the node that is farthest from its correct cluster. The rationale for choosing the node farthest from its correct position is that doing so is most likely the way that a user would select a node. The user would not select a node already close to its correct cluster; she would select the node that was most obviously out of place.

We incorporate our new algorithm in the third and fourth experiments. In the third experiment, we randomly select a node that has not been previously selected and move it towards its correct cluster. After two or more nodes have been moved, we run our new batch growing k-means, with k initially set to two and $maxK$ set to a value of eight. We stop when every node has been moved. In the fourth experiment, again the same steps are performed except that we select the node that is farthest from where it "should" be positioned.

5.3 Experiment Results

In this section, we detail the way we have decided to record how the algorithm is performing during the experiments, as well as what the actual results are. We group our results into sections about each data set, and explain the reasons for the results.

5.3.1 Standard of Success

During the experiments, we recorded three values: the number of cluster centers (the value of k), the percentage of nodes that are in the correct clusters and the percentage of nodes that are in the incorrect clusters, both of which belong to cluster centers which have some node fixed on screen. This metric for judging clustering correctness was not an easy decision, for there is no true standard for judging how correct a clustering is. We hope to provide a comprehensive explanation of how this decision for the metric was made.

Logically, we only have knowledge of what a correct cluster is through user input. Via our user interface, this means which nodes the user has moved. Due to the fact that COP-Kmeans *cannot* violate user constrains, there is no need to judge based on that criterion; we will always have 100 per cent compliance. Therefore, we are only justified in judging the correctness of clusters about which information is known.

By reporting the percentage of nodes that are in the correct cluster as well as the percentage that are misplaced, we get a fairly representative view of how the algorithm is doing overall. Combined with recording the number of cluster centers at each step, we can compare the performance of the algorithm with various values of k. Though this method of judging correctness of a clustering is not flawless, the existing methods are not appropriate to judge an algorithm with a dynamic k.

5.3.2 The Iris Data Set

The iris data set, as stated above, has one class that is quite easy to separate from the other two; in other words, it is linearly separable from the others. The other two classes are not so easily separable, and we can see this reflected in the chart detailing the results of the various experiments. In figure 5.1 we see that COP-Kmeans arrives at the "correct" clustering close to the end, but does in fact get 100 per cent accuracy. When using the farthest first node selection (figure 5.2), COP-Kmeans achieves 100 per cent accuracy after the 112^{th} node moved. However, these experiments are run with the correct value of k and without a correct value, it would never achieve 100 per cent accuracy.

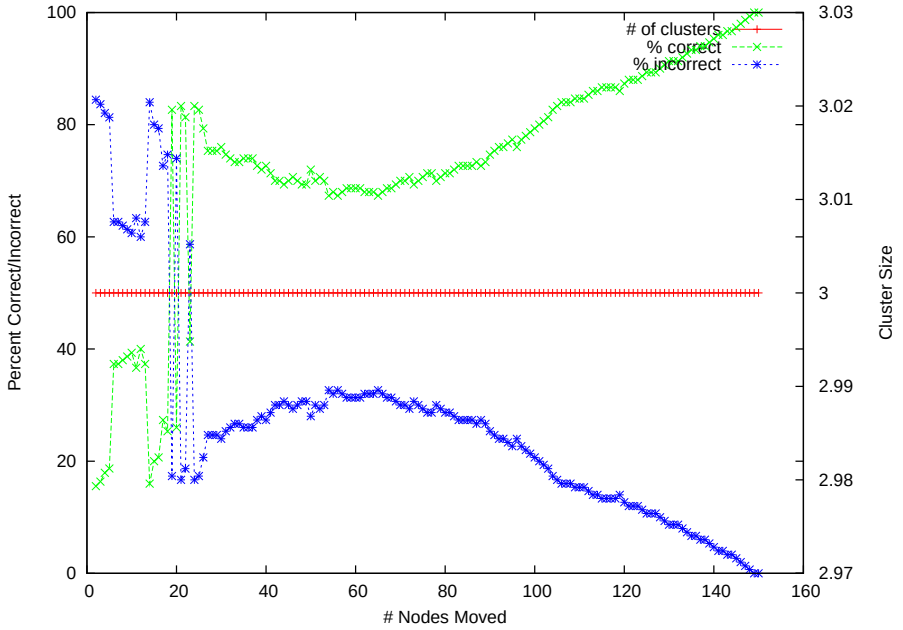

Figure 5.1: COP-Kmeans run on the iris data set with random node selection

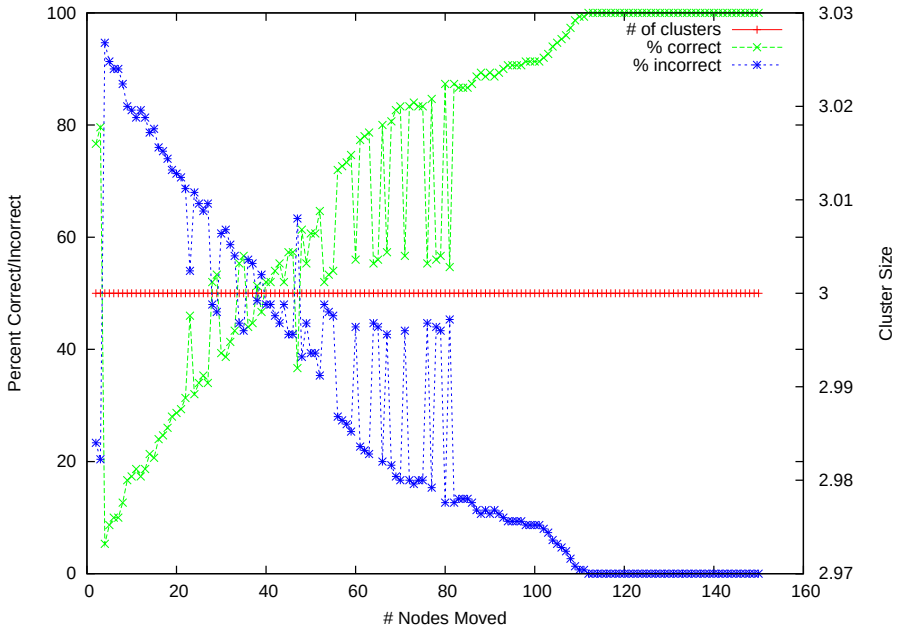

Figure 5.2: COP-Kmeans run on the iris data set with farthest-first node selection

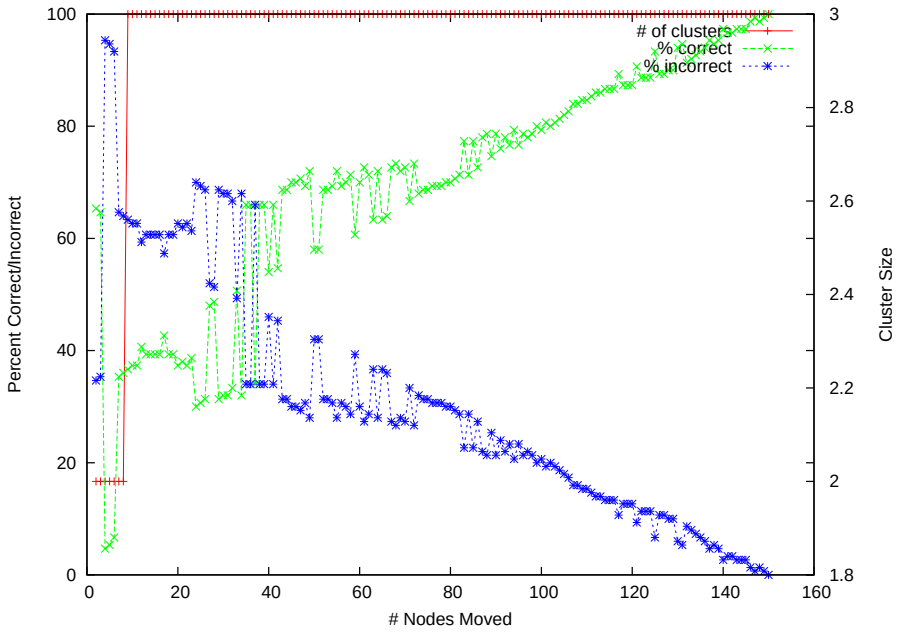

Figure 5.3: Our algorithm run on the iris data set with random node selection

When looking at the charts detailing the performance of our algorithm, we ought to keep in mind one thing: not only is this algorithm's job to classify the data, but it is also attempting to learn the correct value of k. Therefore, we might expect it to learn the correct classifications a little slower than an algorithm with a predefined k. When looking at the classifications for the iris data set though, we have reason to be optimistic. In the experiment with random node selection (figure 5.3), we see that it does learn the correct value of k after only nine nodes were moved. Though it does take until the end to learn the correct clustering, it gets to a reasonably high accuracy prior to that.

However, when using the farthest-first node selection criteria (figure 5.4), we see rather remarkable results. Here it takes nearly twice the time to settle on the correct value of k, but it arrives at 100 per cent accuracy after 119 nodes were moved, and a reasonably high accuracy before that. The farthest-first criteria seems to have a positive result on how quickly the algorithm learns the correct clustering, most likely due to the fact that it corrects the more severe errors early on, before correcting the trivial ones.

5.3.3 The Wine Data Set

The wine data set, like the iris data set, has three classes. However, it is a harder data set to learn as it is a data set of high-dimensionality and the classes are similar.

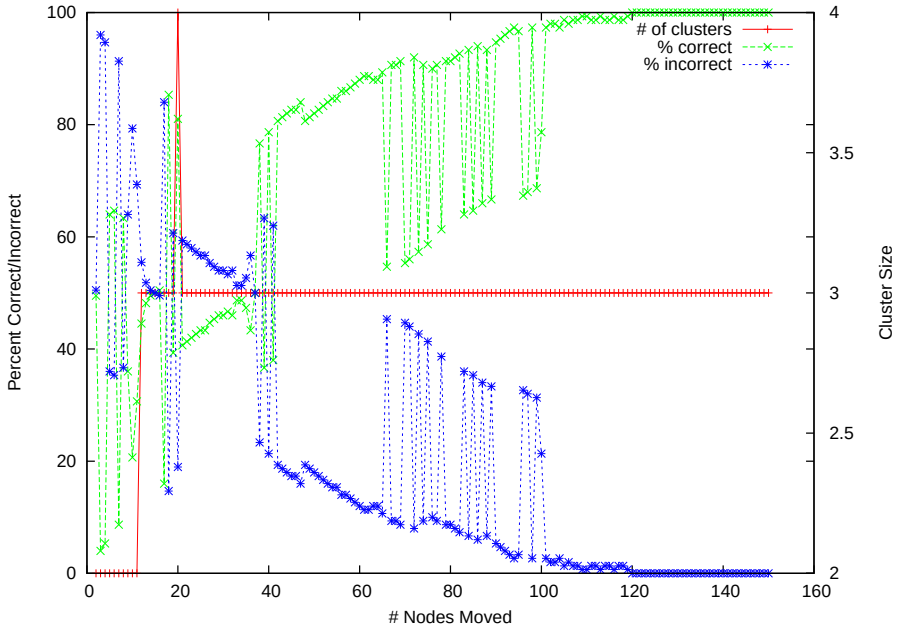

Figure 5.4: Our algorithm run on the iris data set with farthest-first node selection

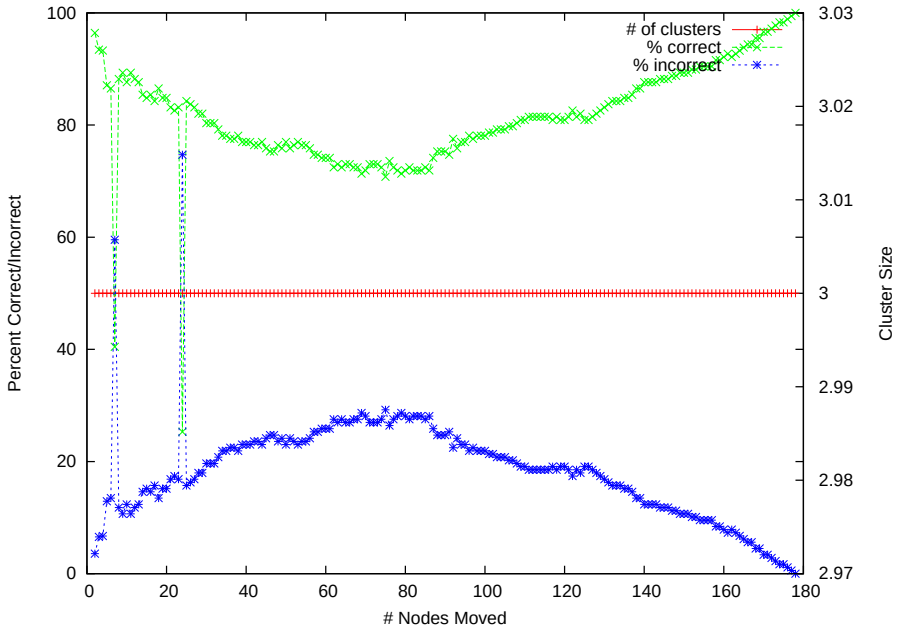

Figure 5.5: COP-Kmeans run on the wine data set with random node selection

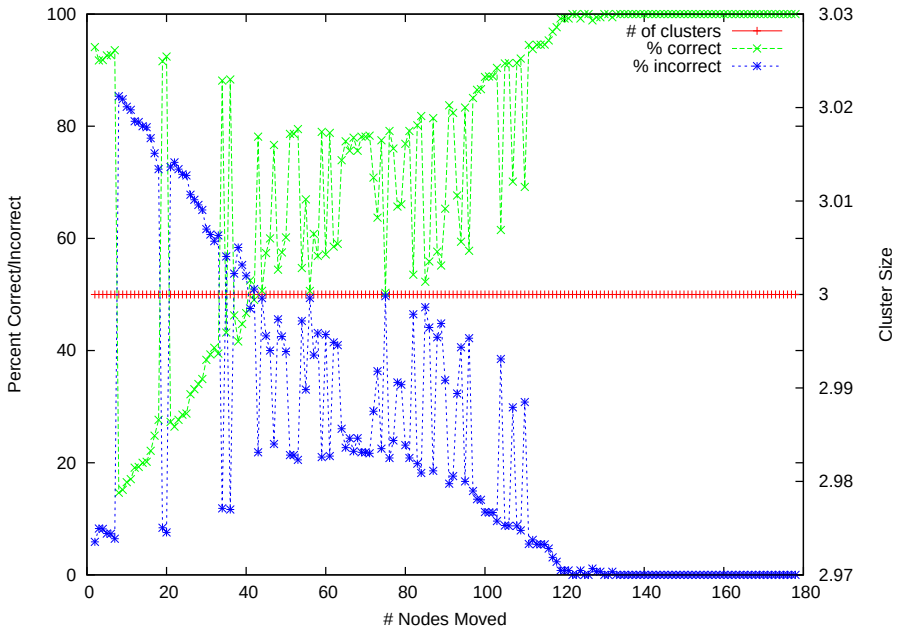

Figure 5.6: COP-Kmeans run on the wine data set with farthest-first node selection

When running COP-Kmeans on it with random node selection (figure 5.5), we get a correct clustering only after all the nodes have been moved to the correct place on screen. Even with a correct value of k, the wine data set is a difficult one to learn.

Yet when the node that is farthest away from its correct cluster is selected (figure 5.6), we see dramatic changes in the results. Instead of taking until the last node moved to get the correct clustering, we see it as early as after the 124^{th} node. It finally settles with 100 per cent accuracy with no subsequent errors after the 132^{th} node is moved.

When running our algorithm, we see less impressive results however. When learning k in the third experiment (figure 5.7), the value goes all the way up to the maximum before coming back down again. It finally settles on the correct value just before the end. The clustering results reach a fairly high accuracy, but reaches 100 per cent accuracy only after all nodes have been relocated.

In the last experiment (figure 5.8), we can see the effect that choosing the farthest node has on the varying value of k. In this instance, k gets to the correct value and stays there; it does not go any higher than three which contrasts sharply with the previous experiment. Here, we get 100 per cent clustering accuracy as early as after the 134^{th} node, but it drops after that and oscillates between the mid 80s and 100 per cent.

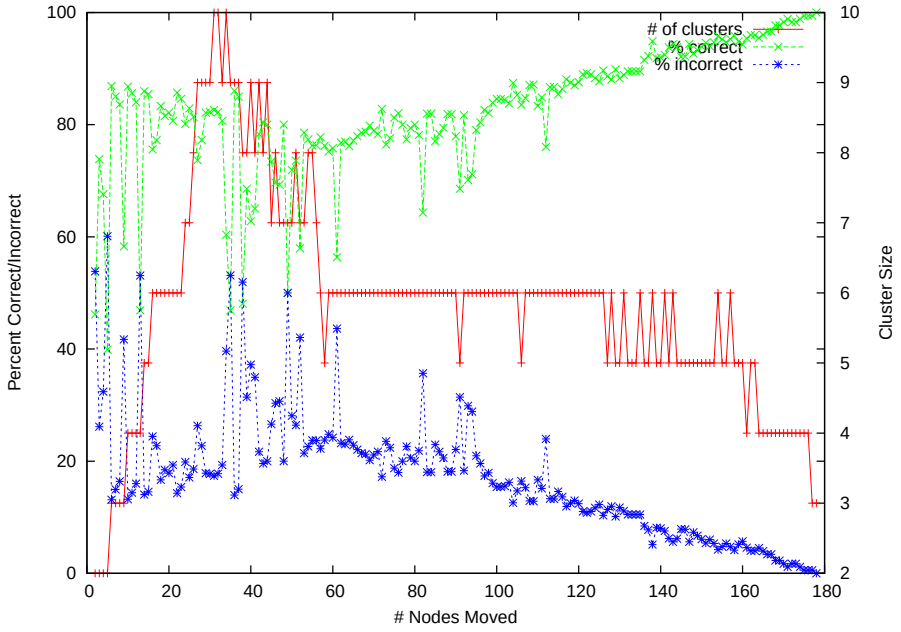

Figure 5.7: Our algorithm run on the wine data set with random node selection

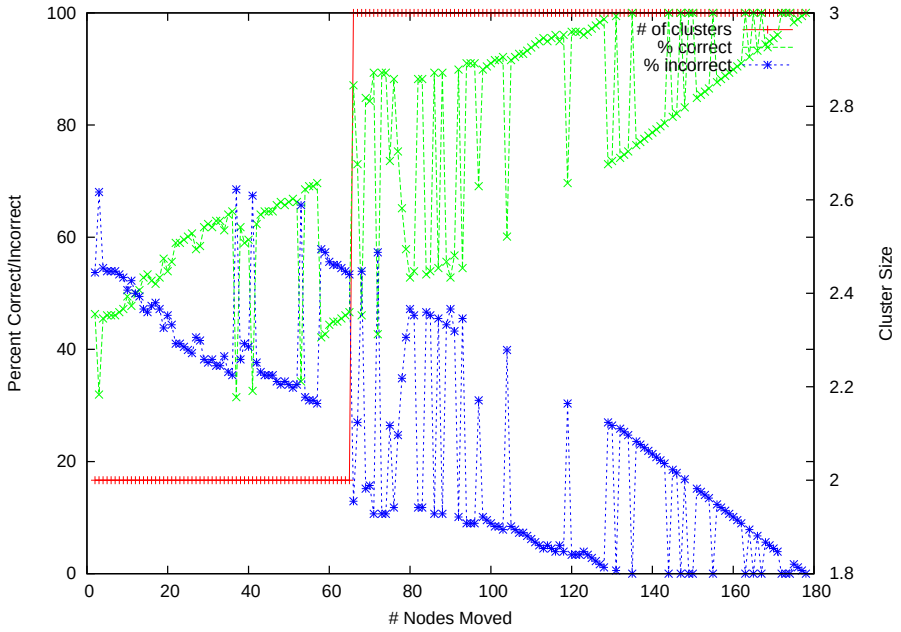

Figure 5.8: Our algorithm run on the wine data set with farthest-first node selection

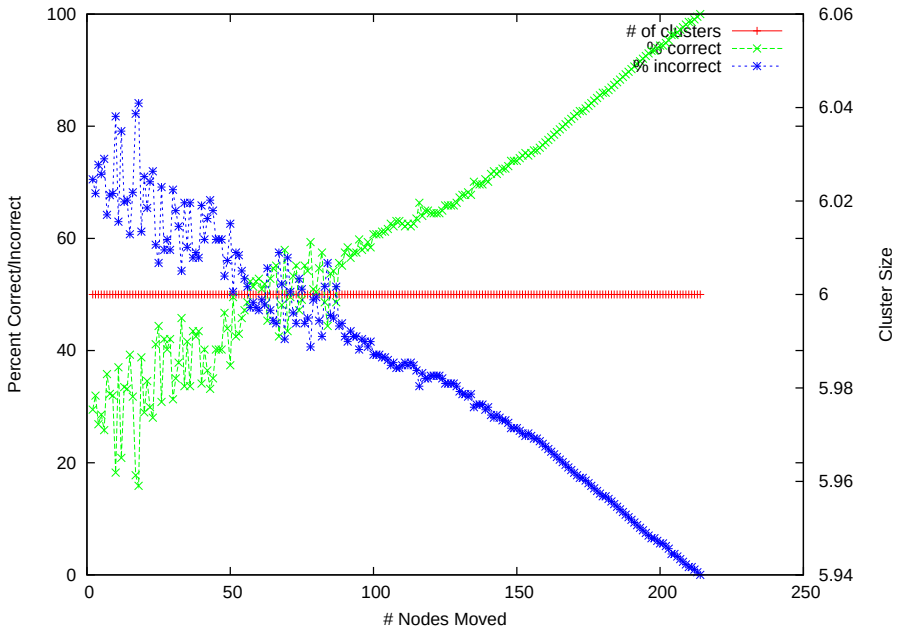

Figure 5.9: COP-Kmeans run on the glass data set with random node selection

5.3.4 The Glass Data Set

The glass data set seems to be the hardest of all to learn with our visual and algorithmic schematic. It has six classes, most of which are not linearly separable from each other. In this case, knowing the correct value of k seems to help some, but not significantly.

By using plain COP-Kmeans with random node selection (figure 5.9), we arrive at the correct clustering only after all nodes have been moved. The percentage of nodes in the correct cluster increases only marginally after each move, which emphasizes the idea that the algorithm is not able to infer about the data set through the set of constraints. When using the farthest-first heuristic (figure 5.10), we don't arrive at the correct clustering significantly faster than with random node selection. There does not seem to be much benefit in changing the node choosing criteria, which separates this data set from the other two.

However, as poorly as just COP-Kmeans does with the glass data set, it is fair to say that our algorithm does even worse. When using random node selection (figure 5.11), we just get by. We arrive at the correct number of clusters quite early on, comparatively, but like COP-Kmeans, we do not arrive at the correct clustering until all nodes have been moved. Though this is passable, the experiment using the farthest-first heuristic completely falls apart during run time. Not only does it not arrive at the correct number of clusters, but the problem becomes over-constrained

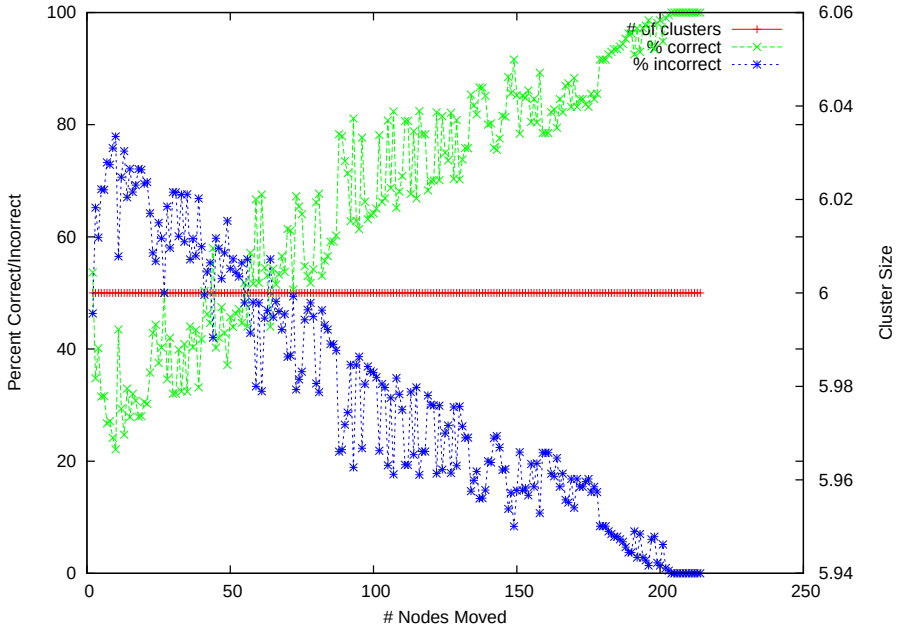

Figure 5.10: COP-Kmeans run on the glass data set with farthest-first node selection

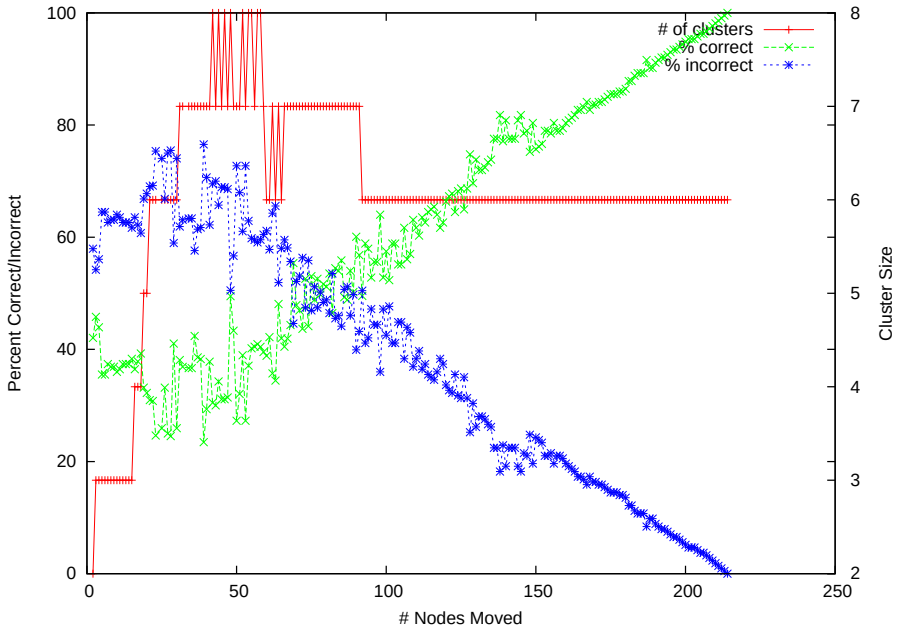

Figure 5.11: Our algorithm run on the glass data set with random node selection

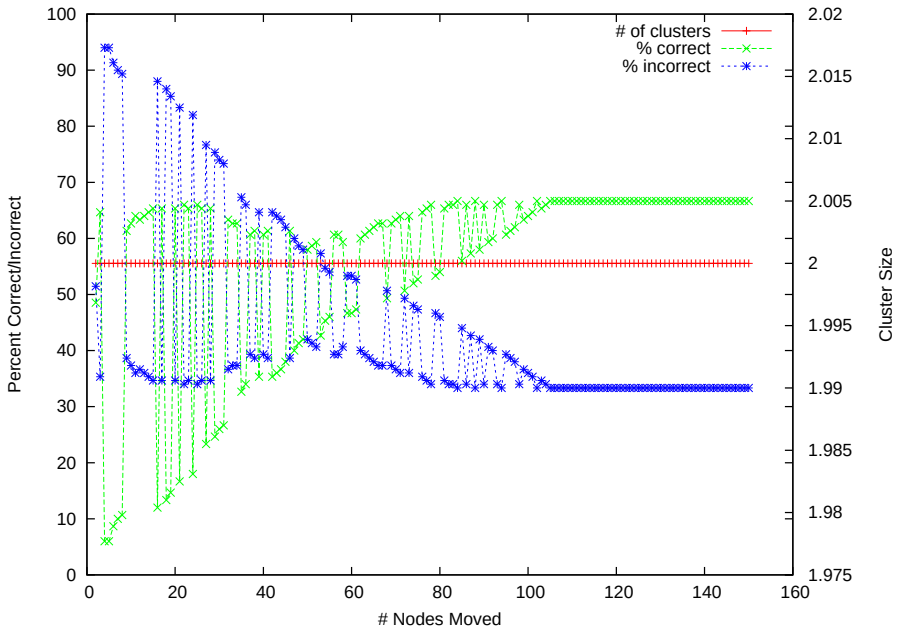

Figure 5.12: COP-Kmeans run on the iris data set with the wrong k

and we get nodes running the screen unattached to any cluster center at all.

5.4 Analysis of Results

In general, no matter what algorithm we are using to classify the data, using the farthest-first choosing criterion seems to be critical to getting the clustering correct sooner. As this seems to be the way that users would interact with the interface anyway, we make no apologies for "faking" the choosing of nodes to move. Also, choosing the one that is farthest from its correct cluster gives the most important feedback, correcting the most glaring errors first. In fact, with an actual user with knowledge of the data set, they would be able to do this even better than choosing solely by distance.

With simple data sets, our algorithm gets to a high percentage of correctness more quickly than COP-Kmeans, and most definitely would without a correct value of k since it can dynamically grow and shrink. When we have data sets of medium or hard difficulty but with a low number of classes, we approximate the performance of COP-Kmeans, which is logical as we are running COP-Kmeans as our clustering algorithm. However, one has to weigh the benefits of a dynamic k versus absolute performance. If we look at a run of COP-Kmeans on the iris data set with a value of k that is too low (figure 5.12), it is obvious how poorly it classifies, due to the fact

that it must get at least an entire class incorrectly classified.

When we have data sets with a high number of cluster centers, it becomes much more difficult for our clustering algorithm. We believe that it is inherent in the visual system: the farthest node will almost always be one in one of the four corners, which will allow the algorithm to settle on a value of k that is not correct but which it thinks is correct. When we find out that another cluster center actually does exist, the algorithm has to readjust, which it does not do perfectly at this point. Ways to avoid this happening will have to be explored.

Chapter 6

Conclusions and Future Work

To the best of our knowledge, there is no existing algorithm that both learns the correct value of k as well as incorporating user constraints. We believe that this research is significant because it explores ways to perform both tasks at the same time. Such an algorithm has applications from alternative ways of searching to medical diagnosis assistance. We can see an algorithm with these capabilities having an enormous impact on existing problems.

6.1 Conclusions about Learning Algorithm and Visualization

Though at this point, our learning algorithm has its weak points, it also shows great potential. For data sets for which no k is known, of for which a k cannot be known, this is a good solution. It may take longer to arrive at an acceptable clustering than one for which a k is known, but this would have to be the compromise for having a dynamic k. In only one experiment was the algorithm not able to determine the correct value, which we believe is an acceptable error rate.

The incorporation of a graphical user interface with the organization of information seems to provide an acceptable level of abstraction while still letting the user know what was going on with each piece of data. Labels help with this, as well as color coding. Visualization of data allows the user to interact with the data on a high level, and effectively be able to classify the data without going through much of the legwork involved in manually classifying data. It lets the user see the entire data set and classify as realizations occur to the user.

6.2 Future Work

There exist several avenues which we would like the opportunity to explore, given time and interest. The first, as was originally included in the problem statement, is a "smart" representation of data. Though the schematic exists, we lacked the time to implement it. Combined with the growing algorithm we can improve upon the "farthest-first" heuristic for choosing nodes to move.

After seeing for ourselves the performance of COP-Kmeans, we would like to see how our algorithm performs using a different classifier, such as PCK-Means or even MPCK-Means. A more flexible clustering algorithm might bring the data to a "correct" clustering more quickly, counter-intuitive though it may seem.

Third, we would like to experiment with different ways to more quickly and more accurately arrive at the correct number of cluster centers (value of k). Whether by experimenting with the method of adding centers, or by fiddling with key variable values, we are confident that the current method can be improved. We believe that changing the methods by which key variables are obtained would also correct the problem encountered with the glass data set. Though we are quite happy with the existing algorithm and visual schematic as a proof of concept, we would welcome the opportunity to refine it.

6.3 Acknowledgments

We would like to thank Dr. Marie desJardins for her inspiration and support, as well as "Doctor" James MacGlashan for paving the way. We would also like to thank the faculty of the Bryn Mawr College Computer Science Department: Dr. Douglas Blank, Dr. Deepak Kumar, Dr. Geoffrey Towell, Dr. George Weaver and Dr. Dianna Xu for all of their help, patience and confidence. It would be remiss of us not to mention the students who shared this lab space, for humoring computer requests as well as random irritability. Special thanks go to Ph.D. candidates Blazej Bulka and Eric Eaton for their help in the initial stages as well as continual reassurance that it is, in fact, worthwhile.

Bibliography

[BBM04] S. Basu, A. Banerjee, and R. Mooney. Active semi-supervision for pairwise constrained clustering. In *Proceedings of the 4th SIAM International Conference on Data Mining*, 2004.

[dFM07] Marie desJardins, Julia Ferraioli, and James MacGlashan. Interactive visual clustering. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*. Intelligent User Interfaces, Jan 2007.

[DNM98] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.

[DWM02] M. Daszykowski, B. Walczak, and D.L. Massart. On the optimal partitioning of data with k-means, growing k-means, neural gas, and growing neural gas. *Journal of Chemical Information and Modeling*, 42(6):1378–1389, 2002.

[HCL05] Jeffrey Heer, Stuart K. Card, and James A. Landay. prefuse: a toolkit for interactive information visualization. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430, New York, NY, USA, 2005. ACM Press.

[KL97] Nandakishore Kambhatla and Todd K. Leen. Dimension reduction by local principal component analysis. *Neural Comput.*, 9(7):1493–1516, 1997.

[LMP01] Neal Lesh, Joe Marks, and Maurizio Patrignani. Interactive partitioning system demonstration, short. In Joe Marks, editor, *Graph Drawing, Colonial Williamsburg, 2000*, pages 31–36. Springer, 2001.

[Mac67] James B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Symposium on Math, Statistics, and Probability*, pages 281–297. University of California Press, 1967.

[MRJ87] A. I. Mees, P. E. Rapp, and L. S. Jennings. Singular-value decomposition and embedding dimension. *Phys. Rev. A*, 36(1):340–346, Jul 1987.

[TMMS93] S. G. Berkovich T. M. Martinez and K. J. Schulten. "neural=gas" network for vector quantization and its applications to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, Jul 1993.

[WCRS01] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schroedl. Constrained k-means clustering with background knowledge. In *Proceedings of 18th International Conference on Machine Learning (ICML-01)*, pages 577–584, 2001.