Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 04-020

A Graphical Interface to Clustering Algorithms and Visualizations

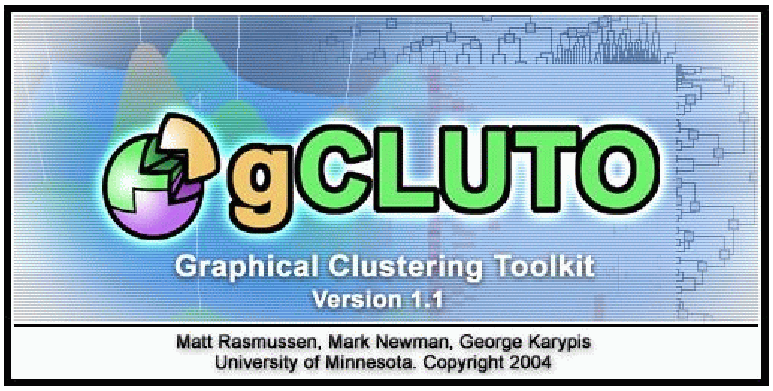
Matthew Rasmussen

May 25, 2004

*g*Cluto: A graphical interface to clustering algorithms and visualizations

Matt Rasmussen

May 25, 2004


gCLUTO
Graphical Clustering Toolkit
Version 1.1
Matt Rasmussen, Mark Newman, George Karypis
University of Minnesota. Copyright 2004

I would like to give special thanks to Professor George Karypis for his guidance and support for this project and my education in general.

Development of *g*Cluto has been funded by the Army High Performance Computing Research Center and National Science Foundation.

I would also like to give credit to Mark Newman for his role in the early development of *g*Cluto.

Contents

Chapter 1

Introduction

Due to recent advances in information technology, there has been an enormous growth in the amount of data generated in fields ranging from business to the sciences. This data has the potential to greatly benefit its owners by increasing their understanding of their own work. However, the growing size and complexity of data has introduced new challenges in extracting its meaning. To address these challenges, many data mining techniques have been developed.

One technique in particular, clustering, has been successful in a wide range of applications. Clustering solves the general problem of identifying groups of related objects. Depending on the application, these objects may represent customers, documents, molecules, or genes. The ability to handle such diverse data in a general way has led to the popularity of clustering algorithms.

In this paper, we introduce *g*CLUTO, a stand alone clustering software package designed to ease the use of clustering algorithms and their results. *g*CLUTO offers improvements over existing tools with features such as an intuitive graphical user interface, interactive visualizations, and mechanisms for comparing multiple clustering solutions. In addition to introducing the tool, the underlining algorithms and design decisions of *g*CLUTO will also be presented.

1.1 Background

Identifying groups of related pieces of information is a common task in data mining that can take on many different forms. In a business situation, for example, a company with a customer sales database may want to identify groups of customers who have similar buying habits in order to understand which demographic groups tend to buy their products. For a small database, this task may be done by inspection. However, as the data increases, simple inspection becomes infeasible and a more systematic method is required.

This same problem also occurs in the field of microbiology [4, 8]. With new technologies such as oligonucleotide chips and cDNA microarrays, researchers can now gather genome-wide information about individual gene expression levels of an organism under varying conditions. Using this information, researchers can infer which genes are co-regulated or are involved in similar biological roles by identifying genes that share similar expression regulation patterns. However, identifying gene groups in microarray experiments is not a trivial task. In most experiments, the data collected contains measurements regarding several thousand genes under varying conditions numbering in the teens.

Fortunately, both of the examples above can be stated in terms of a clustering problem. Given a set of objects, a clustering algorithm attempts to partition the set into groups called clusters, such that objects of high similarity belong to the same cluster and objects of low similarity belong to different clusters. Clustering has been extensively researched in many scientific disciples and many algorithms have been developed [7, 5, 6]. In addition, several software tools have been developed for a wide variety of applications.

A popular toolset for analyzing microarray data is the CLUSTER and TREEVIEW applications written by Michael Eisen at Stanford University [4]. CLUSTER and TREEVIEW are graphical applications that have been integrated together to provide both clustering and visualization abilities. CLUSTER provides agglomerative and k-means clustering, Self-Organizing Maps, and Principal Component Analysis for selecting prominent features. Data can be prepared for clustering using log transforms and normalization. The TREEVIEW application displays the results of CLUSTER as a colored matrix and a hierarchical tree. TREEVIEW provides a few navigation features such as feature searching and zooming.

*g*Cluto offers an alternative to these tools by providing features that make clustering practical for a wide variety of applications. First, *g*Cluto provides an array of clustering algorithms and options through the use of the Cluto clustering library [5]. The Cluto library provides highly optimized implementations of agglomerative, k-means, and graph clustering, especially in the context of sparse high-dimensional data. Second, *g*Cluto helps the user sort through the algorithm options and resulting data files by providing a intuitive graphical interface. Following the paradigm of most development tools, *g*Cluto uses the concept of a "project" in order to organize the user's various datasets, clustering solutions, and visualizations. Lastly, *g*Cluto provides both standard statistics and unique visualizations for interpreting clustering results. Given the wide range of options and factors that are involved in clustering, the user should carefully analyze their results and compare them with results generated with differing options. Therefore, additional effort has gone into visualizations that can facilitate analysis and comparisons.

Chapter 2

Clustering Algorithms

*g*Cluto allows the use of several clustering algorithms by drawing on the algorithms provided by the Cluto clustering library [5]. Cluto supports agglomerative, partitional, and graph partitional clustering algorithms, each of which have different advantages and disadvantages. In the following sections, the Cluto implementation of these algorithms will be explained.

2.1 Agglomerative Clustering

The agglomerative algorithm approaches the clustering problem with a bottom-up perspective. The algorithm begins by placing each object in its own cluster. Next, a series of iterations are performed that successively merges pairs of these clusters into larger clusters. The merge order is guided by a *criterion function* that determines the most advantageous merge in each iteration. The algorithm concludes when either all objects have been merged into one cluster or a specified number of clusters is obtained. The merging process is often visualized by a hierarchical tree, where each node represents a cluster formed by a merge of its children.

The criterion function determines the most advantageous merge by considering the similarities between clusters. Three common functions, single-link, complete-link, and group average, attempt to merge clusters that are most similar. Single-link defines the similarity of a pair of clusters as the maximum similarity of any pair of objects from each cluster. Complete-link, on the other hand, uses the minimum similarity found between any pair of objects. Both of these criterion functions tend to lead to poor results because they use too little information by using only one object to represent an entire cluster [12]. The group average criterion function overcomes this limitation by using the average pair-wise similarity of objects from each cluster.

In addition to the common criterion functions, Cluto also provides several unique functions that approach cluster similarity as an optimization problem [12]. Some of these functions include $\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{E}_1$, $\mathcal{H}_1$, $\mathcal{H}_2$, $\mathcal{G}_1$, and $\mathcal{G}_2$. Functions $\mathcal{I}_1$ and $\mathcal{I}_2$ attempt to evaluate a cluster based on its internal statistics, whereas $\mathcal{E}_1$ focuses on external features such as maximizing the separation between clusters. To simultaneous optimize both internal and external features, one of two hybrid criterion functions, $\mathcal{H}_1$ and $\mathcal{H}_2$, can be used. Lastly, $\mathcal{G}_1$ and $\mathcal{G}_2$

perform cluster optimizations that treat the objects as vertices in a similarity graph.

Although the agglomerative algorithm is a very popular method for clustering, it suffers from several disadvantages, such as a tendency to under optimize the criterion function and large space and time requirements. Agglomerative clustering attempts to optimize the criterion function by greedily choosing merges that are locally optimal, however such merges often lead to a final solution that is suboptimal. The merging process also requires similarity measures to be recalculated for each new cluster that is created. This leads to run-times of $O(n^2 * log(n))$ or $O(n^3)$ for $\mathcal{I}_1$ and $\mathcal{I}_2$ and a space requirement of $O(n^2)$.

2.2 Partitional Clustering

Partitional algorithms offer a good alternative to agglomerative clustering because they can achieve better optimizations for their criterion functions while requiring less time and memory [12]. CLUTO provides two partitional algorithms: direct and repeated-bisection. The direct partitional algorithm performs a k-way clustering by first randomly chosing k objects as *seeds* for k clusters. The remaining objects are then successively added to the cluster that best optimizes the criterion function. After all objects have been assigned to a cluster, random objects are chosen for re-evaluation and are reassigned to a different cluster if it better optimizes the criterion function. The algorithm concludes after a specified number of re-evaluations have occurred.

The advantage of this approach is that poor clustering decisions made early in the algorithm can later be corrected. Also, fewer calculations need to be repeated in each re-evaluation. The partitional approach also addresses the problem of getting caught in a local maximum for its criterion function by using randomness to help explore possibly more optimal clusterings.

The other partitional algorithm, repeated-bisection, works similarly to direct, except only two initial seeds are chosen. Once the objects are partitioned into clusters, the bisection is then repeated separately on each cluster. Bisection repeats until a specified number of clusters is attained. The idea behind this algorithm is that general decisions are optimized before finer tuning occurs. This method contrasts the agglomerative algorithm by performing clustering in a top-down manner.

2.3 Graph Partitional

The last algorithm supported by CLUTO is graph-partitioning-based clustering. Graph partitioning finds clusters with different characteristics than those found by the previous algorithms. The algorithm treats clustering as a graph partitioning problem by constructing a sparse graph, where the objects are represented as vertices and the edges connect objects whose similarity is above a given threshold. The partitioning is performed using a highly efficient multilevel graph-partitioning algorithm [6].

Chapter 3

Using *g*CLUTO

*g*CLUTO is available at *http://www.cs.umn.edu/˜cluto/gcluto* for public download. The software package has been written in C++ and compiled for both Microsoft and Linux platforms. Cross-platform graphics are provided by the wxWindows, OpenGL, and GLUT libraries. The clustering algorithms are implemented with the CLUTO clustering library. CLUTO, written by Professor George Karypis at University of Minnesota, is also available for public download both as a standalone command-line utility and as a C library at *http://www.cs.umn.edu/˜cluto.*

3.1 Clustering Work-flow and Organization

The main strength of *g*CLUTO is its ability to organize the user's data and work-flow in a way that eases the process of data analysis. This work-flow often consists of a sequence of stages, such as importing and preparing data, selecting clustering options, interpreting solution reports, and concluding with visualization. Each stage of the process demands decisions to be made by the user that can alter the course of data analysis. Consequently, the user may want to backtrack to previous stages and create a new branch of analysis. An overview of this work-flow with examples of branching is depicted in Figure 3.1.

*g*CLUTO assists these types of work-flows by introducing the concept of a project. A project manages the various data files, solutions reports, and visualizations that the user generates by storing and presenting them in a single container. Figure 3.2 illustrates how *g*CLUTO uses a tree to represent a project as it progresses through the stages of data analysis.

The work-flow of a user begins by creating a new project. *g*CLUTO will create a new directory to hold all project related files as well as a new empty project tree. Next the user imports one or more related datasets. These datasets are represented by icons that appear directly beneath the project tree's root. After importation, the user can cluster a dataset to produce a clustering solution. For each solution, a solution report is generated which contains statistics about the clustering. Clustering solutions are presented by an 'S' icon and are placed beneath the clustered dataset's icon in the project tree. As more clustering solutions are generated, the project tree will continue to organize them by their corresponding datasets. Lastly, the work-flow concludes with interpreting a solution using one or more visualizations. Again, the project tree will reflect which solutions have generated

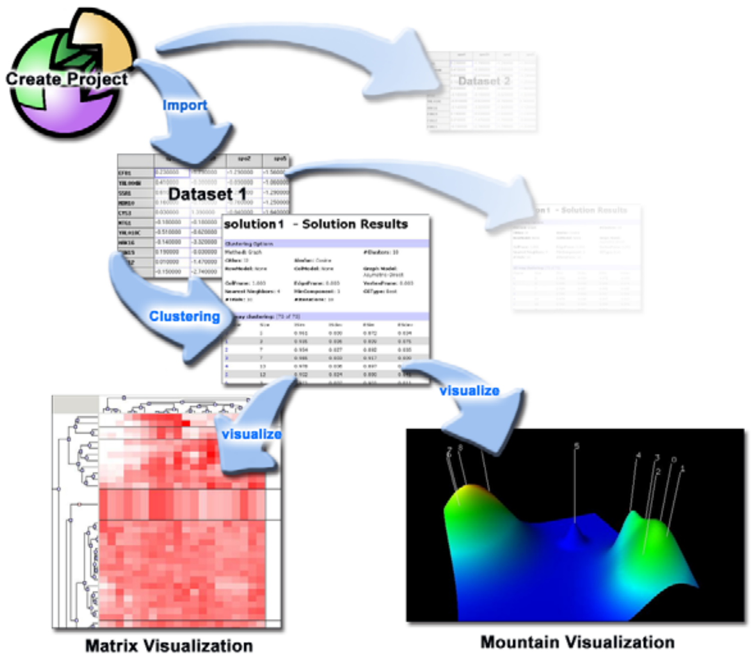


Figure 3.1: Overview of *g*CLUTO's work-flow with example screenshots for each stage

visualizations by placing beneath them visualization icons. In the following sections, we will focus on each stage of the work-flow and introduce how *g*CLUTO assists in data analysis.

3.2 Creating a New Project

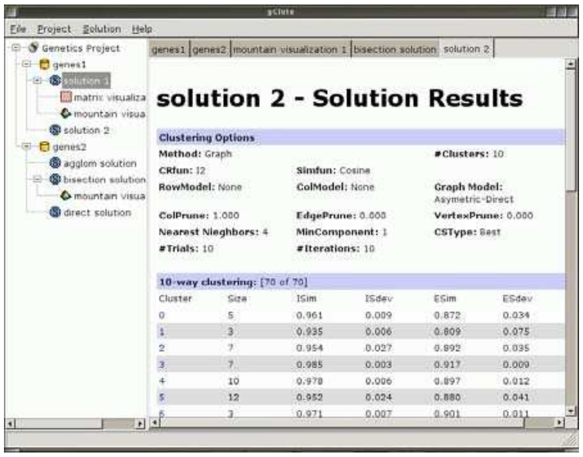
The user begins their analysis by first creating a new project. A project is intended to hold one or more related datasets. The project tree provides an easy interface for switching between datasets and comparing their results.

When a project is saved, all of the project information is saved under a single project directory specified by the user. Within the project directory, directories and text files are used to capture the same tree structure seen in the *g*CLUTO project tree. This straight forward format is used so that third party applications can access *g*CLUTO's project data. In addition, *g*CLUTO allows exporting of solutions and printing of visualizations to standard formats for external use.

3.3 Importing Data

Datasets can be imported into *g*CLUTO in a variety of formats. Currently the supported formats include: CLUTO's vector and similarity formats, and character delimited files.

The vector format contains a matrix written in either dense or sparse form. Each row of the matrix represents an object, whereas each column represents a feature. The value of the



solution 2 - Solution Results

Clustering Options

Method: Graph **#Clusters:** 10
CRfun: I2 **Simfun:** Cosine
RowModel: None **ColModel:** None **Graph Model:** Asymetric-Direct
ColPrune: 1.000 **EdgePrune:** 0.000 **VertexPrune:** 0.000
Nearest Nieghbors: 4 **MinComponent:** 1 **CSType:** Best
#Trials: 10 **#Iterations:** 10

10-way clustering: [70 of 70]

Cluster	Size	ISim	ISdev	ESim	ESdev
0	5	0.961	0.009	0.872	0.034
1	3	0.935	0.006	0.809	0.075
2	7	0.954	0.027	0.892	0.035
3	7	0.985	0.003	0.917	0.009
4	10	0.978	0.006	0.897	0.012
5	12	0.952	0.024	0.880	0.041
6	3	0.971	0.007	0.901	0.011

Figure 3.2: A screenshot of the project tree displaying data items, solutions, and visualizations. On the right is an example of a Solution Report.

of i^{th} row and j^{th} column represents the strength of feature j in object i. With this matrix, *g*CLUTO will compare objects by comparing their vectors.

If such vectors are not available, but information about object pair-wise similarities is available, then the CLUTO similarity format can be used. This format consists of a square matrix with same number of rows and columns as the number of objects. The value in the i^{th} row and j^{th} column represents the similarity of the i^{th} and j^{th} object.

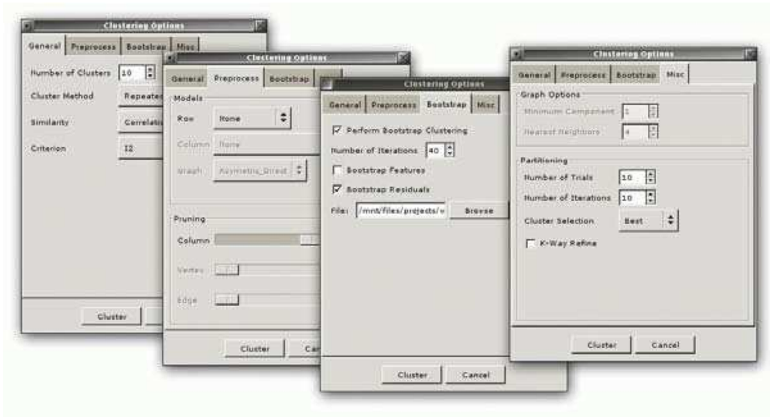
Character delimited files contain the same information as the CLUTO vector format except in a more common and flexible form. Most spreadsheet applications can export data in character delimited formats. The format also allows labels to be present in the first row and column of the matrix.

3.4 Clustering

Once a dataset is imported into *g*CLUTO, clustering can be initiated by selecting the desired options from the clustering options dialog pictured in Figure 3.4. These options have been organized into four sections: *General*, *Preprocess*, *Bootstrap*, and *Miscellaneous*. The most general options include specifying the number of desired clusters, the clustering method, and similarity and criterion functions. The preprocess options allow the user to prepare their data before clustering. This is accomplished by using model and pruning functions. The models scale various portions of the dataset, whereas the pruning options generate a more descriptive subset of the dataset. These options are necessary for datasets that have value distributions that may skew clustering algorithms. See Figure 3.4 for a full listing of all available options. Bootstrap Clustering, another clustering feature provided by *g*CLUTO, is covered in section 3.8.

Clustering Options	
General	
# of Clusters	Number of clusters that the algorithm should find
Method	Underlining clustering algorithm to use
Similarity	Function to measure the similarity between two objects
Criterion	Function to guide algorithm by evaluating intermediate clusterings
Preprocess	
Models	
Row	Scales the values of each row in data matrix
Column	Globally scales the values of each row across rows
Graph	Determines when an edge will exist between two vertices
Pruning	
Column	Remove columns that don't contribute to similarity
Vertex	Remove vertices that tend to be outliers
Edge	Remove edges that tend to connect clusters
Bootstrap	
Perform Bootstrapping	Whether to perform bootstrap clustering
# of Iterations	Number of solutions to create
Bootstrap Features	Whether to resample the features in each iteration
Bootstrap Residuals	Whether to resample data by adding residuals
Miscellaneous	
Graph Options	
Min # of Components	Remove small connected components before clustering
Nearest Neighbors	# of nearest neighbors used in graph-partitioning
Partitioning	
# of Trials	# clusterings to create to search for best clustering
# of Iterations	# of refinement iterations in partitioning
Cluster Selection	Determines how to select next cluster for bisection
K-way refine	Whether to k-way refine a bisection solution

Figure 3.3: Clustering options available in *g*CLUTO. Some options are only available for certain clustering methods.



Number of Clusters: 10
Cluster Method: Repeated
Similarity: Correlati
Criterion: I2
Row: None
Column: None
Graph: Asymetric_Direct
Pruning: Column, Vertex, Edge
☑ Perform Bootstrap Clustering
Number of Iterations: 40
☐ Bootstrap Features
☑ Bootstrap Residuals
File: /mnt/files/projects/v
Graph Options: Minimum Component: 1; Nearest Neighbors: 4
Partitioning: Number of Trials: 10; Number of Iterations: 10; Cluster Selection: Best
☐ K-Way Refine

Figure 3.4: Several screenshots of the clustering dialog

3.5 Solution Reports

Solution Reports are generated for each dataset that is clustered. Solution reports contain information about the clustering options used and statistics about the discovered clusters. These statistics include the number of clusters, cluster sizes, internal and external similarities, internal and external standard deviations, and a list of the most discriminating and descriptive features for each cluster. If known classes are specified for objects, then purity, entropy, and class conservation statistics will also be present.

In Figure 3.5 an example solution report is given for a dataset containing documents about sports. Each object is a document that contains several keywords related to sports. A row class file has been specified for this dataset that allows *g*CLUTO to compare its clustering to the known classes. With this information *g*CLUTO can calculate the purity and entropy of a cluster by noting how many different classes are associated to the objects of the cluster. The class distribution matrix shows how many objects of each cluster belong to each class. From the class distribution, we can see that clusters 0 through 6 associate strongly to a single class. Cluster 7, however, appears to contain objects from many classes.

3.6 Matrix Visualization

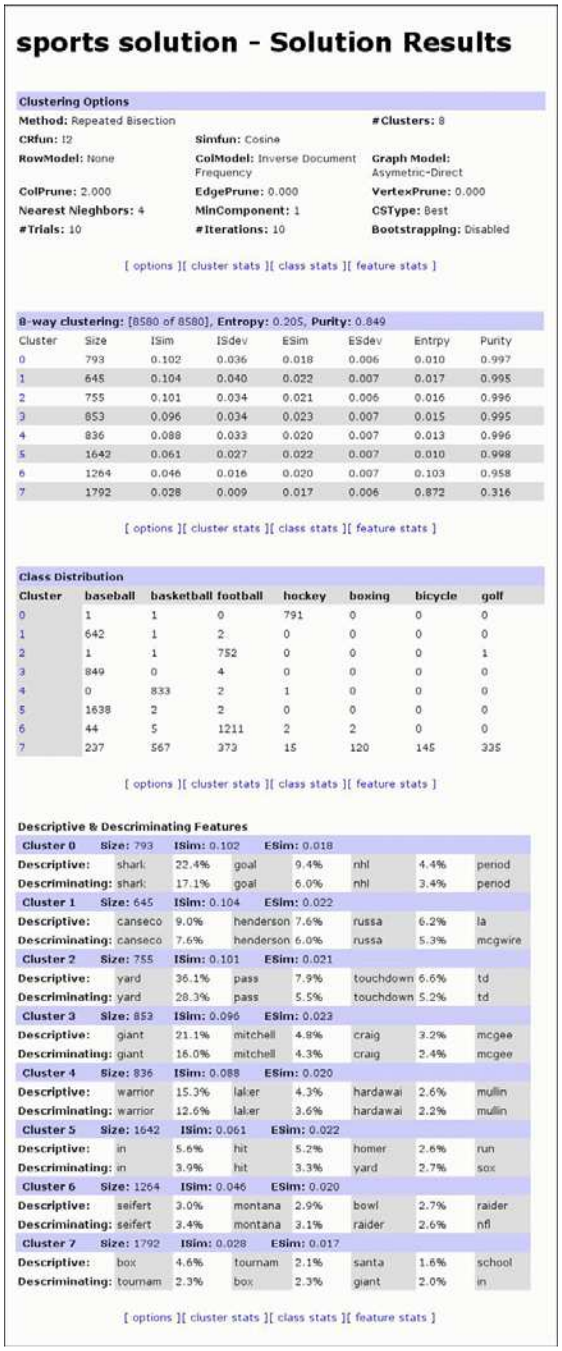
The Matrix Visualization is one of two visualizations that *g*CLUTO provides for analyzing clustering solutions. It is based on visualizations found in many other clustering applications, including CLUTO. However, *g*CLUTO extends existing implementations by providing a completely interactive matrix, with the ability to zoom, query information, and average arbitrary rows and columns.

3.6.1 Interpretation

The Matrix Visualization represents the original data matrix except with a few alterations. First, colors are used to represent the values of the matrix. For example, dark red represents large positive values, while lighter shades are used for values near zero. Conversely, shades of green are used for negative values. Second, the rows of the matrix are reordered in order to display the clusters found during clustering. Objects of the same cluster have their rows placed consecutively and black horizontal lines separate rows belonging to different clusters.

This display allows the user to visually inspect their data for patterns. In an ideal clustering solution, rows belonging to the same cluster should have relatively similar patterns of red and green. The visualization emphasizes these patterns for the user by displaying them in contiguous blocks. If the features represent a sequence, for example measurements in a time-course experiment, then the user can identify trends that occur across the features. The user may also be able to identify more questionable clusters by observing stark dissimilarities between rows within a cluster.

In addition to the color matrix, the visualization also includes labels and hierarchical trees located at the edges of the matrix. If the user supplies labels with their data, then the rows of the matrix will be labeled with object names and the columns with feature names. If the user clusters their data with an agglomerative algorithm, then the agglomerative tree



sports solution - Solution Results

Clustering Options

Method: Repeated Bisection		**#Clusters:** 8
CRfun: I2	**Simfun:** Cosine	
RowModel: None	**ColModel:** Inverse Document Frequency	**Graph Model:** Asymetric-Direct
ColPrune: 2.000	**EdgePrune:** 0.000	**VertexPrune:** 0.000
Nearest Nieghbors: 4	**MinComponent:** 1	**CSType:** Best
#Trials: 10	**#Iterations:** 10	**Bootstrapping:** Disabled

[options][cluster stats][class stats][feature stats]

8-way clustering: [8580 of 8580], **Entropy:** 0.205, **Purity:** 0.849

Cluster	Size	ISim	ISdev	ESim	ESdev	Entrpy	Purity
0	793	0.102	0.036	0.018	0.006	0.010	0.997
1	645	0.104	0.040	0.022	0.007	0.017	0.995
2	755	0.101	0.034	0.021	0.006	0.016	0.996
3	853	0.096	0.034	0.023	0.007	0.015	0.995
4	836	0.088	0.033	0.020	0.007	0.013	0.996
5	1642	0.061	0.027	0.022	0.007	0.010	0.998
6	1264	0.046	0.016	0.020	0.007	0.103	0.958
7	1792	0.028	0.009	0.017	0.006	0.872	0.316

[options][cluster stats][class stats][feature stats]

Class Distribution

Cluster	baseball	basketball	football	hockey	boxing	bicycle	golf
0	1	1	0	791	0	0	0
1	642	1	2	0	0	0	0
2	1	1	752	0	0	0	1
3	849	0	4	0	0	0	0
4	0	833	2	1	0	0	0
5	1638	2	2	0	0	0	0
6	44	5	1211	2	2	0	0
7	237	567	373	15	120	145	335

[options][cluster stats][class stats][feature stats]

Descriptive & Descriminating Features

Cluster 0	**Size:** 793	**ISim:** 0.102	**ESim:** 0.018				
Descriptive:	shark	22.4%	goal	9.4%	nhl	4.4%	period
Descriminating:	shark	17.1%	goal	6.0%	nhl	3.4%	period
Cluster 1	**Size:** 645	**ISim:** 0.104	**ESim:** 0.022				
Descriptive:	canseco	9.0%	henderson	7.6%	russa	6.2%	la
Descriminating:	canseco	7.6%	henderson	6.0%	russa	5.3%	mcgwire
Cluster 2	**Size:** 755	**ISim:** 0.101	**ESim:** 0.021				
Descriptive:	yard	36.1%	pass	7.9%	touchdown	6.6%	td
Descriminating:	yard	28.3%	pass	5.5%	touchdown	5.2%	td
Cluster 3	**Size:** 853	**ISim:** 0.096	**ESim:** 0.023				
Descriptive:	giant	21.1%	mitchell	4.8%	craig	3.2%	mcgee
Descriminating:	giant	16.0%	mitchell	4.3%	craig	2.4%	mcgee
Cluster 4	**Size:** 836	**ISim:** 0.088	**ESim:** 0.020				
Descriptive:	warrior	15.3%	laker	4.3%	hardawai	2.6%	mullin
Descriminating:	warrior	12.6%	laker	3.6%	hardawai	2.2%	mullin
Cluster 5	**Size:** 1642	**ISim:** 0.061	**ESim:** 0.022				
Descriptive:	in	5.6%	hit	5.2%	homer	2.6%	run
Descriminating:	in	3.9%	hit	3.3%	yard	2.7%	sox
Cluster 6	**Size:** 1264	**ISim:** 0.046	**ESim:** 0.020				
Descriptive:	seifert	3.0%	montana	2.9%	bowl	2.7%	raider
Descriminating:	seifert	3.4%	montana	3.1%	raider	2.6%	nfl
Cluster 7	**Size:** 1792	**ISim:** 0.028	**ESim:** 0.017				
Descriptive:	box	4.6%	tournam	2.1%	santa	1.6%	school
Descriminating:	tournam	2.3%	box	2.3%	giant	2.0%	in

[options][cluster stats][class stats][feature stats]

Figure 3.5: Example solution report of a clustering of sports related documents. The sections of this report in order are clustering options, cluster statistics, class distribution, and descriptive and discriminating features.

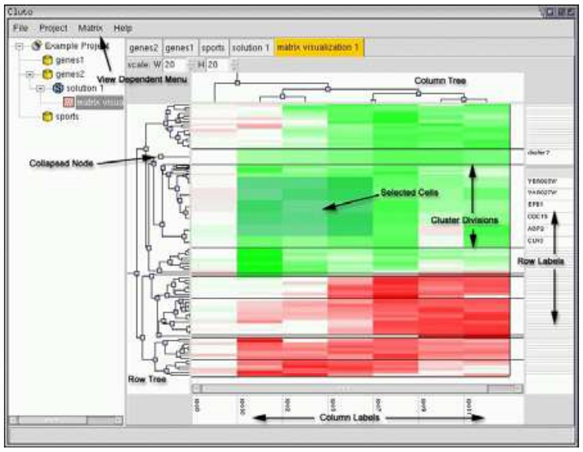


Figure 3.6: A screenshot of the Matrix Visualization

will be displayed on the left-hand side of the visualization. The user may also generate a hierarchical tree even if a partitional clustering algorithm was used. In such cases, *g*CLUTO performs additional agglomerative clustering within each partitional cluster and a single agglomerative clustering of the clusters themselves. Using the trees generated from these additional clusterings, *g*CLUTO constructs a single hierarchical tree that conforms to the same cluster structure found with the partitional algorithm. Lastly, the Matrix Visualization can also display a hierarchical tree called the feature tree, which is generated by performing agglomerative clustering on the transpose of the data matrix.

Similar to visualizations in other clustering applications, the hierarchical tree depicts relationships between objects by displaying the order in which objects were merged in the agglomerative process. Since merging is performed by descending pair-wise similarity, objects that are near each other in the tree are more similar than objects placed in distant locations. However, if users want to draw conclusions about object similarities using the hierarchical tree, they must keep in mind that a two-dimensional drawing of a hierarchical tree is not unique. That is, for every parent node in the tree, the two children nodes and their subtrees can be drawn in one of two possible orientations: top or bottom (note: *g*CLUTO draws the hierarchical tree with children placed to the upper-right or lower-right of their parents). *g*CLUTO removes this ambiguity by explicitly ordering the visual position of each subtree by choosing the set of orientations that maximizes the similarity between objects placed in consecutive rows in the Matrix Visualization.

3.6.2 Manipulating the Matrix Visualization

Once the Matrix Visualization is generated, users can further explore their results by manipulating the visualization in several ways. First, the user may collapse any set of rows or columns in the matrix by collapsing the corresponding nodes in the hierarchical trees located above and to the left of the matrix. By collapsing a node of the tree, the user can hide all of the node's descendants. In the matrix, the corresponding rows that belong to the leaves of the collapsed sub-tree are replaced by a single representative row. The representative row contains the average vector of all of the hidden rows and, thus, summaries the data in a condensed form. This feature is especially useful for large datasets that are difficult to fully display on a computer monitor. Columns can also be collapsed in a similar manner. When a representative row crosses a representative column, the intersection is a representative cell, which contains the average value of the cells contained within the collapsed rectangular region.

A frequent use of row averaging is to view the cluster mid-point vectors. This can be done either by collapsing the appropriate nodes in the object hierarchical tree, or by selecting the "Show Only Clusters" option from the "Matrix" menu. The user may also quickly expand all collapsed nodes by choosing the "Show All Objects" option from the "Matrix" menu.

The last manipulation that is available to the user is scaling. A common problem with viewing similar visualizations in other applications, is that it is difficult to represent a large dataset on a relatively small display. One solution is to only display a portion of the visualization at any one time and allow the user to scroll to view other portions. The downside to this solution is that the user has a narrow view of their data, which makes it difficult to compare local details to the global trends. Another solution is to shrink the graphics until they fit within the viewable area. In cases where the matrix has more rows and columns than the number of pixels available, it becomes difficult to appropriately represent the matrix without excessive distortion. *g*CLUTO implements a unique compromise by allowing the user to zoom in on portions of the matrix that are of interest, while zooming away from portions that are less important but are still needed for context.

Scaling is initiated by selecting a rectangular region of cells in the matrix by dragging the mouse. Once selected, the rectangular region can be scaled to a larger or smaller size by using the mouse and dragging on any of the region's edges. This action will rescale the selected region, while keeping the scaling of neighboring regions intact. The visualization also provides several menu options and controls for performing common scalings.

3.7 Mountain Visualization

The Mountain Visualization is another visualization that *g*CLUTO provides for analyzing a clustering result.

3.7.1 Purpose

The purpose of the Mountain Visualization is to visually aid the user in understanding the contents of a high-dimensional dataset. The dimension of a dataset is problem specific and is determined by the number of features present, which can be on the order of tens to thousands. Since it is not convenient to directly display this data on a two-dimensional screen, a function must be chosen that maps the high-dimensional data to an easily displayed lower-dimensional representation. For each cluster, the Mountain Visualization provides the number of constituent objects, internal similarity, external similarity, and standard deviation. The visualization attempts to summarize all of this information into one graphical form.

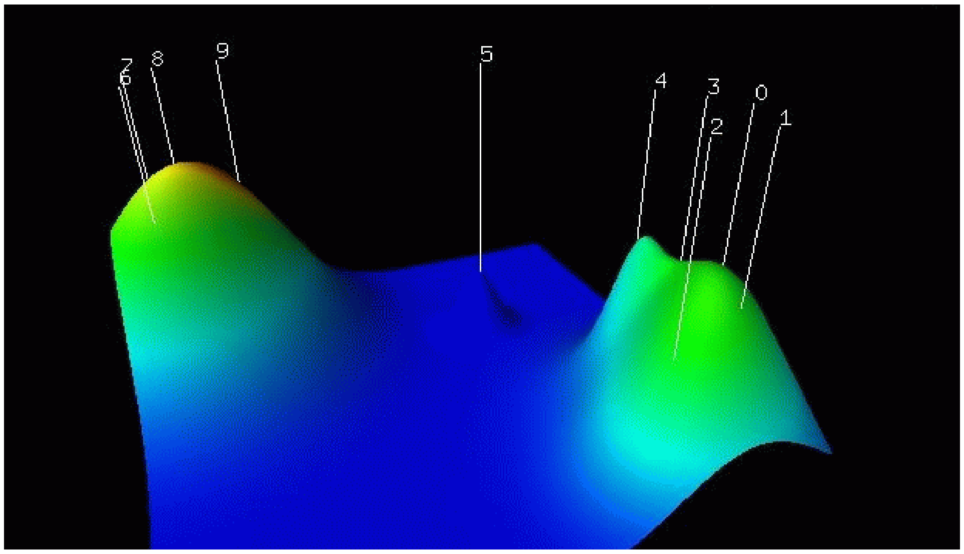


Figure 3.7: A screenshot of the Mountain Visualization displaying ten clusters in two major groups

3.7.2 Interpretation

When a user generates a Mountain Visualization from a clustering result, a 3D OpenGL window displaying a colored mountain-like terrain is opened. This terrain consists of a horizontal plane which rises in peaks in several locations. Each peak represents a single cluster in the clustering. Information about the corresponding cluster is represented by the peak's location on the plane, height, volume, and color.

The most informative attribute of a peak is its location on the plane with respect to other peaks. The distance between a pair of peaks on the plane represents the relative similarity of their clusters. Similarity is measured using the same similarity function chosen for clustering. The purpose of this representation, is to illustrate the relationships between clusters using visual distance. In this manner, clusters that are similar will have peaks that lie closely together, whereas more dissimilar clusters will be displayed with distant peaks.

In Figure 3.7, an example Mountain Visualization is given of a clustered dataset. Although this clustering specifies ten clusters, the visualization has chosen to place these peaks into two large groups. This indicates a more general structure in the dataset, namely two large dissimilar clusters with high internal similarity. Given this information, the user may make conclusions about the meaning of their clusters or may chose to re-cluster their data with a different specified number of clusters.

The height of each peak on the plane is proportional to the internal similarity of the corresponding cluster. Internal similarity is calculated by finding the average pair-wise similarity between objects of the cluster. The volume of a peak is proportional to the number of objects within the cluster. Lastly, the color of a peak represents the internal standard deviation of the cluster's objects. Red represents low deviation, whereas blue represents high deviation. The internal deviation is calculated by finding the average standard deviation of the pair-wise similarities between the cluster's objects.

The overall effect of this representation is to emphasize features of highly structured data. For example, the user will be able to quickly identify clusters with high similarity by finding tall peaks. Also the user will be able to identify clusters with low standard deviation, another feature of structured data, by finding peaks with "hot" colors, such as red and orange. Clusters with high standard deviation are often "noisy" and so they are given a cool blue color. Since the default color of the terrain is blue, noisy and less meaningful clusters appear to blend into the background.

While the visualization uses many features to portray information, it also contains other features which do not have any significance in representing the data. For example, each peak is represented by a Gaussian curve. This curve is used to approximate the distribution of the objects within the cluster, but it does not guarantee that such a distribution exists. To create a smooth terrain, the Gaussian curves are added together. For distant peaks, this addition is negligible, however for closely placed peaks the addition increases the resultant height of each peak. The user should keep this addition in mind when comparing heights between peaks. Lastly, only the color appearing at the tip of a peak is significant. At all other locations, the color is determined by blending in order to create a smooth transition.

3.7.3 Implementation

The core algorithm behind the Mountain Visualization is the mapping function between the original high-dimensional dataset and the two-dimensional data that is displayed. The Mountain Visualization uses Multidimensional Scaling (MDS) to find a mapping that minimizes the data's distortion as it is mapped to the plane.

MDS is an algorithm that takes as input a list of high-dimensional vertices and outputs a list of lower-dimensional vertices. In *g*CLUTO's implementation, the cluster midpoints are used as input and the output consists of two-dimensional points, which are used to place peaks on the plane of the visualization. MDS evaluates a particular mapping using a stress function that calculates the mapping's distortion using a sum-of-squared-error calculation. The optimal mapping is defined as the mapping with the least error, which is found by:

$$Error = \frac{1}{\sum_{i<j} \delta_{i,j}} \sum_{i<j} \frac{(d_{i,j} - \delta_{i,j})^2}{\delta_{i,j}} \tag{3.1}$$

Where $d_{i,j}$ is the distance between two lower-dimensional vertices, $\mathbf{y}_i$ and $\mathbf{y}_j$, and $\delta_{i,j}$ is the distance between their higher-dimensional counterparts, $\mathbf{x}_i$ and $\mathbf{x}_j$. $d_{i,j}$ is found using Euclidean distance. However, the $\delta_{i,j}$ is found by using the clustering's chosen similarity function. This design decision was made so that relationships found between clusters were based on the same method of comparison as used for deriving the internal contents of the clusters.

To search the space of possible mappings for the optimum mapping, a gradient-descent procedure is used [3]. First an initial mapping is made by projecting the cluster midpoints to the plane using the two dimensions that produce the least error. Next, the mapping is iteratively improved by using the gradient given in (3.2) to adjust the lower-dimensional vertices. Gradient-descent is executed until either adjustments are relatively small or a maximum fixed number of iterations occur.

$$\nabla_{\mathbf{y}_k} Error = \frac{2}{\sum_{i<j} \delta_{i,j}} \sum_{j \neq k} \frac{d_{k,j} - \delta_{k,j}}{\delta_{k,j}} \frac{\mathbf{y}_k - \mathbf{y}_j}{d_{i,j}} \tag{3.2}$$

3.8 Bootstrap Clustering

Although the clustering problem is well defined as an optimization problem, there can still be uncertainty associated with its result if the input dataset contains uncertainty. For example, in the case of clustering data generated from measurements, each measurement will have a margin of error. Without additional analysis, a clustering algorithm will cluster the data under the assumption that the data is completely accurate. However, it would be more appropriate if the algorithm could incorporate the uncertainties associated with the data to produce a clustering result that could portray its level of statistical significance given the uncertainties.

Bootstrap clustering is a technique introduced by [8] that adds the statistical technique of bootstrapping to clustering algorithms. Bootstrapping simulates the multiple sampling of a distribution by randomly selecting values from a known sample with replacement. By sampling with replacement, new hypothetical datasets can be produced from the original dataset that exhibit the same distribution of values and uncertainties. This allows clustering algorithms to explore what results would occur if the same measurements were taken again.

*g*CLUTO implements two methods of bootstrapping data: resampling features and resampling residuals. To resample the features of a dataset, *g*CLUTO randomly selects with replacement columns from the dataset to produce a new set of features. This resampling tests to what extent the clustering algorithm may be relying on any particular feature. The resampling of residuals is performed by first supplying a residual matrix for the dataset. A residual matrix contains the errors associated with a dataset, which can be found by fitting the data to a linear model. In [8], residuals for microarray data are found by fitting the data to an ANOVA model. *g*CLUTO can accept residual matrices stored in character delimited file formats. With the residual matrix, *g*CLUTO performs bootstrapping to generate a new residual matrix, which is then added to the original dataset to produce a new hypothetical dataset.

Bootstrap clustering uses these hypothetical datasets to estimate the significance of a clustering solution by clustering each hypothetical dataset and comparing all of their clustering solutions. *g*CLUTO provides three statistics for reporting a clustering solution's significance: solution stability (3.4), cluster stability (3.3), and object stability (3.5). The term stability refers to the level of consistency observed between the various clustering results. These stability measurements range from zero (no consistency between solutions) to one (complete consistency between solutions). Solution stability represents the significance of the solution as a whole, where as cluster and object stability portray a significance level on a per cluster and per object basis.

Before solutions can be compared, a mapping must be made between their partition labels. Clustering labels the input objects with a set of labels to denote to which cluster

```
$Bootstrap$(data, $n$ = number of bootstrap iterations)
**let** $s$ = array of $n$ solutions
**for** $i$ = 1 to $n$
    **let** $d$ = $Resample$($data$)
    **let** $s[i]$ = $Cluster$($d$)
**end for**
**let** $d^*$ = $CreateSimilarityGraph$($s$, $n$)
**let** $s^*$ = $Cluster$($d^*$)
**for** $i$ = 1 to $n$
    **let** $c$ = $FindConfusionMatrix$($s^*$, $s[i]$)
    **let** $m$ = $FindMapping$($c$)
    **let** $stabilityStatistics$ = $AccumulateStabilityStatistics$($s^*$, $s$, $m$)
**end for**
**output** $s^*$, $stabilityStatistics$
```

Figure 3.8: Overview of bootstrap algorithm

each object belongs. *g*CLUTO uses the set of numbers ranging from zero to the number of clusters minus one. Although, each clustering in the bootstrapping uses the same set of labels, there is no guarantee that the cluster labeled x in one solution will have the same labeling in another. In addition, the identification of the "same" cluster in two different solutions can be difficult if the cluster has slightly different members in each solution. A mapping between the labels of one solution to another resolves these problems.

A label mapping is defined as the mapping that maximizes the consistency of the object labelings between two solutions. This mapping is found by first calculating the *confusion matrix* between two solutions. The confusion matrix is a NxN matrix where N is the number of clusters and the value in the i^{th} row and j^{th} column represents the number of objects that are in the i^{th} cluster of solution 1 and the j^{th} cluster of solution 2. The optimal mapping is the then the selection of N cells from the confusion matrix such that no cell shares a column or a row and the sum of the selected values is maximized. This problem can be solved in polynomial time using the Hungarian algorithm.

One disadvantage of the mapping technique described above is that it can only facilitate the comparison of two solutions at a time. In bootstrap clustering, we must compare several solutions. To accomplish this, *g*CLUTO finds a *consensus solution* to which a mapping is found to all other solutions. This star-like mapping arrangement allows comparisons to be made between any pair of solutions while also requiring the fewest mappings to be found.

Lastly, the only task left to be addressed is the generation of the consensus solution. The consensus solution is found by generating a solution that is most similar to most of the solutions. To produce a consensus solution, we can cluster the objects using a similarity graph built from information about the many bootstrap solutions. *g*CLUTO builds a similarity graph by defining the similarity of two objects as the percentage of bootstrap solutions that assign the two objects to the same cluster. The consensus solution is also the final solution that *g*CLUTO presents to the user in the solution report.

3.8.1 Stability Calculations

The following stability statistics are given in gCluto.

Cluster Stability of cluster i in s^*

$$Stability(s^*.C_i) = \frac{1}{|s^*.C_i||S|} \sum_{s_j \in S} \sum_{o \in s_j.C_i} (s_j(o) == s^*(o)) \tag{3.3}$$

Solution Stability of solution s^*

$$Stability(s^*) = \frac{1}{|O|} \sum_{s^*.C_i \in s^*} Stability(s^*.C_i)|s^*.C_i| \tag{3.4}$$

Object Stability of object o

$$Stability(o) = \frac{1}{|S|} max_{c \in s^*.C} \left\{ \begin{array}{l} \text{Number of assignments of } o \\ \text{to cluster } c \text{ across bootstraps} \end{array} \right\} \tag{3.5}$$

Where the == operation has the following meaning

$$(x == y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

and the variables have the following definitions

Set of objects	O
Set of features	F
Set of solutions in bootstrapping	S
Consensus solution	s^*
Cluster id of object o in solution s	$s(o)$
Set of objects in cluster i in solution s	$s.C_i$

Chapter 4

Software Design

*g*CLUTO began as a 2002 summer research project for the Army High Performance Computing Research Center Summer Institute under the guidance of Professor George Karypis at the University of Minnesota [9]. During that time, I collaborated with another intern, Mark Newman. I continued development the following fall as a Undergraduate Research Assistant for Karypis and produced an initial beta release, *g*CLUTO 0.5, on Feburary 17, 2003. By November 27, 2003, *g*CLUTO 1.0 was released providing greater stablity and feature completeness. Most recently, I have worked on *g*CLUTO 1.1 as part of my Computer Science Honors Thesis. This section will cover the software design and organization as of *g*CLUTO 1.1.

4.1 Design Goal

The design goal of *g*CLUTO has been to provide an easy to use cross-platform data clustering and visualization tool. The target user may or may not be knowledgeable about the subtleties of clustering, therefore *g*CLUTO has been designed to provide reasonable defaults for users who want quick results, while providing power users with access to all configuration options involved in clustering.

4.2 Feature Set

*g*CLUTO provides nearly all of the features of CLUTO plus the addition of several unique features.

4.3 Software Organization

*g*CLUTO currently has five layers to its design. In increasing dependency, they are the libgcluto base library, Project Items, Controls, Views, and the MainFrame. These layers allow for code reuse, flexibility, and organization. These layers are illustrated in Figure 4.2.

Version	*Features*
1.1	• Dialogs updated and stored in XML resource file
	• Projects can open from command-line
	• Reduced disk space requirements for projects
	• Object/Feature search in Data View and Matrix Vis
	• Data Properties Dialog
	• Bootstrap Clustering
1.0	• Reorganized project directories
	• Files are deleted when corresponding items are deleted from project
	• Long operations performed in a background thread
	• Importing tab, space, etc. delimited files into projects
	• Printing and Exporting data and visualizations
	• Exporting data/solution matrix to tab delimited files
	• Exporting Solution Report to HTML file
	• Printing Matrix and Mountain Visualizations to printers and files
	• Solution columns in Data View
	• Sorting in Data View
	• External Clustering Quality Statistics in Solution Reports
	• Right click information in Matrix Visualization
	• Matrix Visualization labels stretch with available space
	• "View All Objects" and "View Only Clusters" added to Matrix Visualization
	• Mini-Solution Report in Mountain Visualization
0.5	• Initial beta release
	• Support for Windows and Linux platforms
	• Project Tree for organization
	• Dialogs for choosing clustering options
	• Spreadsheet interface for viewing data
	• HTML interface for viewing solutions
	• Matrix Visualization - a colored interactive matrix
	• Mountain Visualization - a 3D visualization generated using Multidimensional Scaling.

Figure 4.1: Feature history of *g*CLUTO

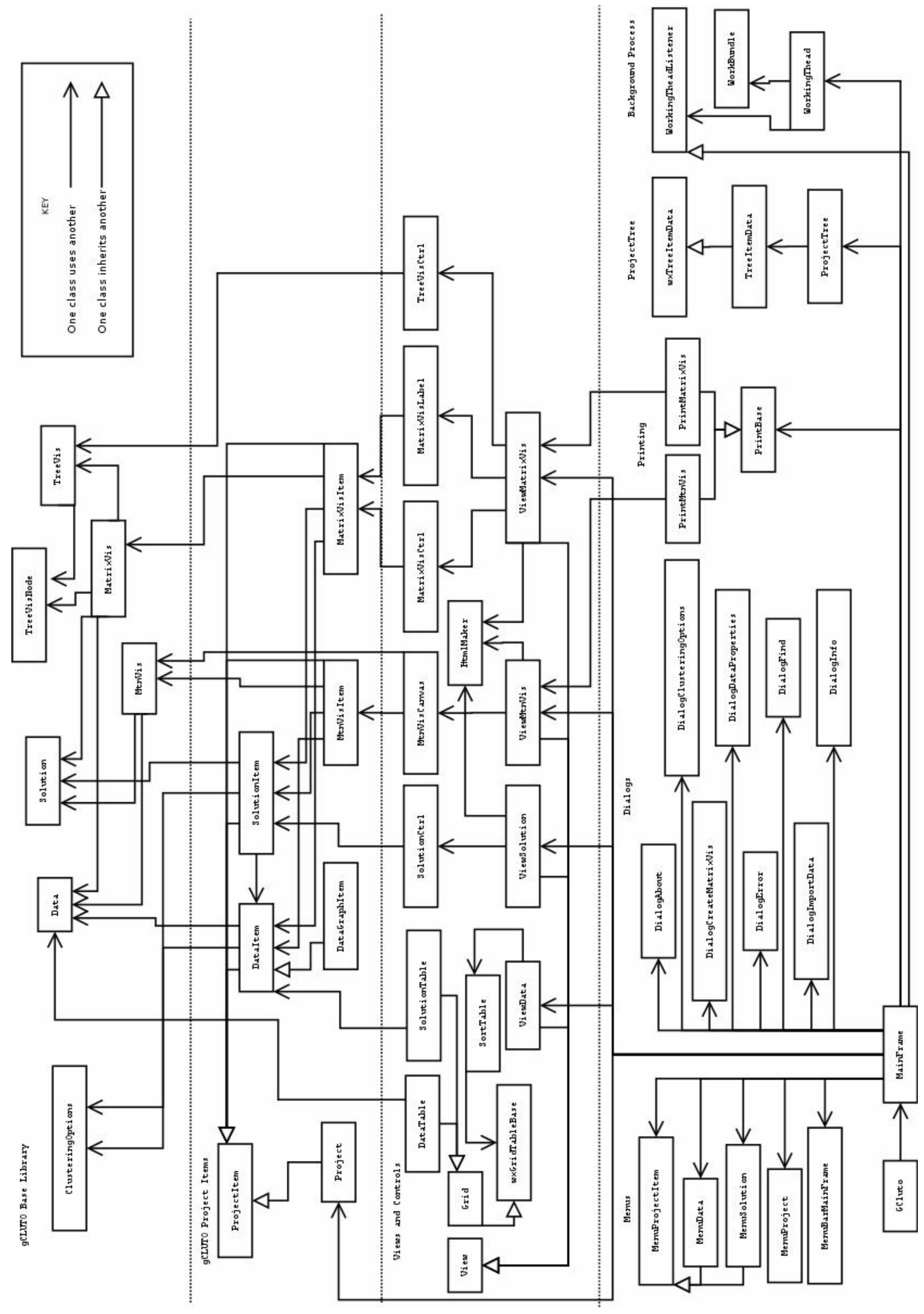


Figure 4.2: Class diagram of *g*Cluto and its base library, libgcluto

4.3.1 libgcluto - Base Library

The role of the libcluto is to provide object-orientation wrappers for CLUTO's algorithms and any additional algorithms used in *g*CLUTO. libgcluto is purely a backend library (no graphical components) and so can be used in a variety of applications. Its main classes are Data, Solution, ClusteringOptions, MatrixVis, MtnVis, TreeVis, TreeVisNode. libcluto also provide convenient global functions for common tasks.

Data

The Data class is the object-oriented version of CLUTO's matrix structure. Data encapsulates the matrix (in sparse or dense form) and optional labels such as the row labels, column labels, and class labels. Data provides algorithms such as transposition (dense and sparse), internalization (applying CLUTO preprocessing options to data), submatrix creation from selected features, and reading and writing of data to and from several file formats.

Solution

The Solution class is the object-oriented version of *g*CLUTO's part, ptree, tsims, and gains arrays. Solution's main method is Cluster() which accepts a Data and ClusteringOptions object and populates Solution's data structures with the resulting clustering solution. Solution takes care of selecting the proper CLUTO function and arguments for clustering by inspecting the Data and ClusteringOptions objects. Solution can also produce solution statistics such as cluster and feature statistics which are represented by the ClusterStats and FeatureStats objects. Reading and writing of solution files can be performed by the Solution object. Lastly, Solution contains algorithms for performing bootstrap clustering. Bootstrapping is performed by enabling bootstrapping in the ClusteringOptions object passed to the Cluster() method.

ClusteringOptions

The ClusteringOptions class is a very passive class that merely encapsulates all of the possible options for clustering data. ClusteringOptions does have one method, Init(), which is used to initialize the object to work with a particular type of Data object. Since the class is so small and passive, most member variables are public and are accessed directly by the objects that use ClusteringOptions.

MatrixVis

The MatrixVis class represents the core data structures of Matrix Visualization. This class provides methods for generating a Matrix Visualization from a Data and Solution pair as well as methods for manipulating the matrix: collapsing and expanding the hierarchical trees and scaling cells. Since MatrixVis is a completely backend object, it contains no methods for displaying its data structures. Instead other classes must be designed (See MatrixVisCtrl) to display the MatrixVis by querying it for details about its contents. This design allows the basic MatrixVis algorithms to be reused in displaying the Matrix Visualization in possibly different ways.

In addition, the MatrixVis has been designed to optionally work with the TreeVis class to represent the two hierarchical trees in the Matrix Visualization. When a visualization is generated with hierarchical trees, MatrixVis queries the TreeVis objects about the state of their collapsed and expanded nodes. MatrixVis does not manipulate the TreeVis objects directly though. This design was chosen to keep the MatrixVis and TreeVis classes independent and to allow possibly new ways of combining the two objects in visualizations. The apparent coordination between the two objects in the Matrix Visualization is actually performed by ViewMatrixVis object which captures the user's interaction with the visualization and updates the MatrixVis and TreeVis accordingly.

MtnVis

The MtnVis class plays a role similar to the MatrixVis class in that it represents the Mountain Visualization but does not provide any methods for displaying the visualization. This class is used by *g*CLUTO to create an OpenGL visualization, whereas gcluto-batch uses MtnVis to produce VRML. MtnVis makes use of the Multidimensional Scaling algorithm implemented in MDS.cpp.

TreeVis

The TreeVis class represents the hierarchical trees used in the Matrix Visualization, although the class could also be used to create a visualization of its own. In addition to storing a hierarchical tree, TreeVis also stores information about collapsing and expanding nodes and the display widths of the nodes. Display widths are used in the Matrix Visualization when cells are scaled and the hierarchical trees must be adjusted to match. TreeVis also provides methods for attaining common collapsed node states, such as viewing all objects or viewing all clusters.

TreeVisNode

The TreeVisNode class is a small passive class used by the TreeVis class to store information about each node in the hierarchical tree.

4.3.2 Project Items

Although libgcluto provides object-oriented representations for the most common data structures in *g*CLUTO, the classes do not provide enough information for *g*CLUTO to operate. What is primarily missing from the libgcluto library is the concept of a project. This concept is not implemented in the libgcluto library because applications that use the library may want to organize their work-flow in a different way. Therefore, the project item layer in *g*CLUTO has been designed to add the concept of a *g*CLUTO project to each class in the libgcluto library. These classes include: ProjectItem, Project, DataItem, DataGraphItem, SolutionItem, MatrixVisItem, and the MtnVisItem. These classes are also completely back-end and provide no code for displaying their contents.

ProjectItem

The ProjectItem class is an abstract base class for all other project items. Some of its responsibilities are to serve as a node in the project tree, provide a common interface to loading and saving a ProjectItem, and provide a mechanism for calling back objects that would like to "listen" for updates to the ProjectItem. The tree maintained by project items is also completely independent of the wxWindows tree widget used to display the project tree. The only interaction that a ProjectItem has with the project tree widget is to store a wxTreeItemId for the widget. ProjectItem assumes the widget will maintain this id.

Project

Although its name does not end in "Item", the Project class is a subclass of the ProjectItem class. Project represents the root of the project tree. Its main purpose is to provide a recursive methods for loading and saving an entire project. The saving method is smart enough to know not to save child ProjectItems that have not changed since the last load. DataGraphItem is a subclass of DataItem specialized for similarity graphs.

DataItem and DataGraphItem

The DataItem and DataGraphItems are project aware versions of the libgcluto Data class. DataItem forwards common data manipulations such as transposition and internalization onto its internal Data object. Also DataItem provides implementations for the ProjectItem Load and Save interface methods that take care of the saving of data and labels into the proper files. Lastly, DataItem assists in interpreting the import data options and calling the proper lower level methods to accomplish a data import.

SolutionItem

The SolutionItem works much the same as the DataItem. It forwards common queries to the encapsulated Solution object and implements the loading and saving methods for storing a SolutionItem in a project. One additional complexity is that the SolutionItem has the ability to save an internalized Data object in its directory. After clustering a DataItem the SolutionItem creates an internalize version of the data to be used in visualizations. This internalized data can be saved for use in future *g*CLUTO sessions. If the internalized data is not saved, then it is regenerated from the original DataItem (the SolutionItem's parent) during loading.

MatrixVisItem and MtnVisItem

These Items are currently very small. Besides forwarding the visualization generation method, they do nothing other than maintain a node in the project tree. Loading and saving of visualization has not been implemented yet in *g*CLUTO.

4.3.3 Controls

Control classes implement the display and user interface of *g*CLUTO's backend objects. Controls collect requests for action from wxWindows's event system, interprets them, and manipulates its encapsulated backend object accordingly. Controls make no assumptions about any possibly surrounding controls or their parent window. This design allows possibly new ways of combining controls to create interfaces. To coordinate controls that must cooperate, such as the MatrixVisCtrl, MatrixVisLabel, and TreeVisCtrls do in the Matrix Visualization, controls provide callback abilities using a "listener" convention (as often used in Java GUI programming). These callbacks are serviced by the parent View window that then manipulates all of its controls such that they stay in sync. See the ViewMatrixVis class for an example of this coordination.

DataTable and SolutionTable

The DataTable and SolutionTable are *g*CLUTO's customized spreadsheet controls for displaying data matrices and the solution columns in the Data View. Both tables accept a Data or Solution object to display and can work with the SortTable to provide sorting abilities. The DataTable has the additional ability to display sparse matrices.

SolutionCtrl

The SolutionCtrl provides the display of the Solution Report. SolutionCtrl is subclassed from wxHtmlWindow. Its main responsibility is to service link click events in the HTML. These links allow the user to quickly jump between related sections of the report.

MatrixVisCtrl

The MatrixVisCtrl displays the actual colored matrix in the Matrix Visualization. The control is responsible for catching user clicks and drags on the matrix and performing the appropriate selecting and scaling actions. Whenever scaling or scrolling occurs, the MatrixVisCtrl calls back its parent window through a generic interface, MatrixVisCtrlListener. This window is then responsible for propagating these events on to neighboring controls, such the TreeVisCtrl and MatrixVisLabel.

MatrixVisLabel

This control simply displays either the row or column labels of a Matrix Visualization. It can be manipulated by the parent window to scroll, show a subset of labels, or scale various labels.

TreeVisCtrl

The hierarchical trees in the Matrix Visualization are handled by the TreeViewCtrl. The control catches mouse clicks to collapse and expand the nodes of its tree. These actions are propagated by callbacks to its parent window which it then responsible for updating neighboring controls, such as the MatrixVisCtrl and MatrixVisLabel. The TreeVisCtrl can be manipulated to adjust scrolling and scaling of its tree.

MtnVisCanvas

The MtnVisCanvas is a subclass of the wxGLCanvas class and provides the OpenGL display of the Mountain Visualization. It reads the height and color matrices stored in the MtnVis object and executes the proper GL commands to render the visualization. It also provides user navigation by catching mouse clicks and drags.

4.3.4 Views

When a user clicks a project item in the project tree, a window called a View is opened to display the project item. These views contain one or more controls in order to display its project item. Views often listen to callbacks from their controls in order to respond to actions originating from them. Views are also responsible for creating and deleting their controls. When a user wants to export or print a project item, it is the corresponding View that performs the exporting or printing. Lastly, Views provide menus and toolbars to allow further user interaction.

View

The View class provides a generic interface for the MainFrame to open, close, and update Views. All Views are subclassed from the View class.

ViewData

The ViewData class implements the Data View. It coordinates the synchronization of the DataTable and SolutionTable controls. It can export its information as a character delimited file containing the data matrix, labels, and solution columns.

ViewSolution

The SolutionView simply provides exporting abilities and a window for the SolutionCtrl. When exporting is chosen, the SolutionView copies the underlining HTML of the SolutionCtrl to a file.

ViewMatrixVis

The ViewMatrixVis is the window of the Matrix Visualization. It coordinates the synchronization of the MatrixVisCtrl, MatrixVisLabel, and TreeVisCtrl. The View also optionally creates labels and trees depending on whether labels are present or whether the user has enabled trees.

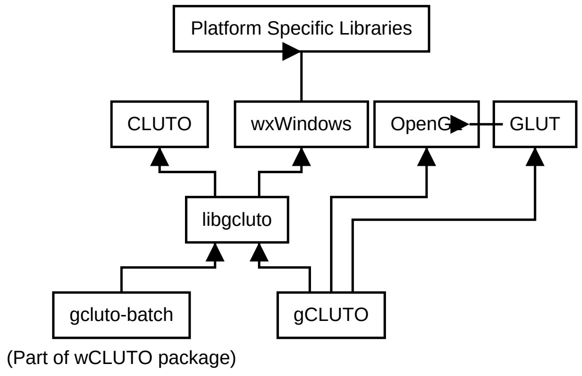


Figure 4.3: Diagram of *g*CLUTO's library dependencies

ViewMtnVis

The Mountain Visualization window is implemented by the ViewMtnVis class. The ViewMtnVis class combines the MtnVisCanvas and an HTML control to create the Mountain Visualization.

4.3.5 *g*CLUTO's Main Frame

The final layer of *g*CLUTO's design is the MainFrame class which coordinates objects from all of the previous layers. This giant class is responsible for displaying the project tree, servicing menu events, initiating actions, printing, displaying dialogs, and managing Views. However, despite its size and broad responsibilities, most of it methods simple route information from one source to another.

4.3.6 Library Dependencies

Figure 4.3 illustrates the library dependencies of *g*CLUTO and related projects such as *w*CLUTO. CLUTO provides the underlining clustering algorithms for *g*CLUTO[5]. wxWindows provides cross-platform interfaces to graphics toolkits, file operations, and thread control [11]. wxWindows in turn depends on additional libraries specific to the platform (Windows or Linux). OpenGL [2] provides the Mountain Visualization's three dimensional graphics and GLUT [1] provides three dimensional text rendering. libgcluto is the backend of *g*CLUTO and contains object-oriented wrappers for CLUTO's algorithms as well as additional algorithms used by *g*CLUTO. gcluto-batch is an example of another application using the libgluto library. gcluto-batch is a command line only utility created for *w*CLUTO (Web Enabled Clustering Toolkit) [10] that generates VRML (Virtual Reality Modeling Language) representations of the Mountain Visualization.

4.4 Conclusions and Future Work

*g*CLUTO has been designed as a general tool for clustering and visualizing data. Although efforts have been made to ease the process, it remains mainly a tool for technically knowledgeable researchers. One consideration for future work is to specialize *g*CLUTO for specific problem domains so that researchers in those domains could more easily interact with the tool. Example problem domains would be analysis of microarray or document databases. In a microarray setting, *g*CLUTO could be customized to accept common microarray formats. In addition, microarray specific visualizations and outputs could be added. For document databases, *g*CLUTO could assist in the generation of the initial data matrix from a database of documents or from an online query.

One of *g*CLUTO's latest features is bootstrap clustering. Bootstrap clustering performs comparisons between bootstrap solutions in order to produce significance statistics. A possible extension of this idea would be to allow the user to compare any arbitrary set of solutions in a similar manner. For example, the user may want to find the stability of solutions found using different algorithms or differing clustering options. In addition to producing stability statistics, a new visualization could be developed to visually compare and contrast differing solutions.

Bibliography

[1] Glut. http://www.sgi.com/software/opengl/glut.html.

[2] Opengl. http://www.opengl.org.

[3] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*, volume November. John Wily & Sons, Inc., New York, second edition, 2000.

[4] Michael B. Eisen, Paul T. Spellman, Patrick O. Brown, and David Botstein. Cluster analysis and display of genome-wide expression patterns. *pnas*, 95:14863–14868, Dec 1998.

[5] George Karypis. Cluto—a clustering toolkit, 2002. http://www.cs.umn.edu/˜cluto.

[6] George Karypis and Vipin Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1999.

[7] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[8] M. Kathleen Kerr and Gary A. Churchill. Bootstrapping cluster analysis: Assessing the reliability of conclusions from microarray experiments. *Proceedings of the National Academy of Sciences (PNAS)*, 98(16):8961–8965, July 2001.

[9] Matt Rasmussen. Ahpcrc: gcluto graphical frontend project, 2002. http://www.ahpcrc.org/education/archives/2002/mrasmuss/.

[10] Matthew D. Rasmussen, Mukund S. Deshpande, George Karypis, James Johnson, John A. Crow, and Ernest F. Retzel. wcluto: A web-enabled clustering toolkit. *Plant Physiology*, 113:510–516, October 2003.

[11] Julian Smart. wxwindows - cross platform toolkit. http://www.wxwindows.org.

[12] Ying Zhao and George Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In Konstantinos Kalpakis, Nazli Goharian, and David Grossmann, editors, *Proceedings of the Eleventh International Conference on Information and Knowledge Management (CIKM-02)*, pages 515–524, New York, November 4–9 2002. ACM Press.