CLUTO *

A Software Package for Clustering High Dimensional Datasets

Release 1.5

George Karypis

University of Minnesota, Department of Computer Science
Minneapolis, MN 55455

karypis@cs.umn.edu

January 8, 2002

 This work was supported by NSF CCR-9972519, EIA-9986042, ACI-9982274, by Army Research Office contract DA/DAAG55-98-1-0441, by the DOE ASCI program, and by Army High Performance Computing Research Center contract number DAAH04-95-C-0008. Related papers are available via WWW at URL: **http://www.cs.umn.edu/~karypis**. The name CLUTO is derived from CLUstering TOolkit.

Contents

1 Introduction **4**
- 1.1 What is CLUTO . . . 4
- 1.2 When CLUTO Should be Used . . . 4
- 1.3 Outline of CLUTO's Manual . . . 5

2 Major Changes From Release 1.0 **6**

3 Using CLUTO via its Stand-Alone Program **7**
- 3.1 The vpcluster Clustering Program . . . 7
  - 3.1.1 Optional Parameters . . . 7
- 3.2 The vacluster Clustering Program . . . 14
  - 3.2.1 Optional Parameters . . . 15
- 3.3 Understanding the Information Produced by vpcluster and vacluster . . . 16
  - 3.3.1 Internal Cluster Quality Statistics . . . 17
  - 3.3.2 External Cluster Quality Statistics . . . 17
  - 3.3.3 Looking at each Cluster's Features . . . 18
  - 3.3.4 Looking at the Hierarchical Agglomerative Tree . . . 18
  - 3.3.5 Looking at the Visualizations . . . 23
- 3.4 Input File Formats . . . 25
  - 3.4.1 Matrix File . . . 25
  - 3.4.2 Row Label File . . . 26
  - 3.4.3 Column Label File . . . 26
  - 3.4.4 Row Class Label File . . . 29
- 3.5 Output File Formats . . . 29
  - 3.5.1 Clustering Solution File . . . 29
  - 3.5.2 Tree File . . . 29

4 CLUTO's Library Interface **31**
- 4.1 Using CLUTO's Library . . . 31
- 4.2 Matrix Data Structure . . . 31
- 4.3 Clustering Parameters . . . 32
  - 4.3.1 The *simfun* Parameter . . . 32
  - 4.3.2 The *crfun* Parameter . . . 32
- 4.4 Object Modeling Parameters . . . 32
  - 4.4.1 The *rowmodel* Parameter . . . 32
  - 4.4.2 The *colmodel* Parameter . . . 33
  - 4.4.3 The *colprune* Parameter . . . 33
- 4.5 Debugging Parameter . . . 33
- 4.6 Clustering Routines . . . 34
  - CLUTO_VP_ClusterDirect . . . 34
  - CLUTO_VP_ClusterRB . . . 35
  - CLUTO_VA_Cluster . . . 36
  - CLUTO_SA_Cluster . . . 37
  - CLUTO_V_BuildTree . . . 39
- 4.7 Cluster Statistics Routines . . . 41
  - CLUTO_V_GetSolutionQuality . . . 41
  - CLUTO_V_GetClusterStats . . . 42
  - CLUTO_V_GetClusterFeatures . . . 44

CLUTO_V_GetTreeStats 46
CLUTO_V_GetTreeFeatures 47

5 System Requirements and Contact Information **49**

6 Copyright Notice and Usage Terms **49**

1 Introduction

Clustering algorithms divide data into meaningful or useful groups, called *clusters*, such that the intra-cluster similarity is maximized and the inter-cluster similarity is minimized. These discovered clusters can be used to explain the characteristics of the underlying data distribution and thus serve as the foundation for various data mining and analysis techniques. The applications of clustering include characterization of different customer groups based upon purchasing patterns, categorization of documents on the World Wide Web, grouping of genes and proteins that have similar functionality, grouping of spatial locations prone to earth quakes from seismological data, *etc.*

1.1 What is CLUTO

CLUTO is a software package for clustering high dimensional datasets and for analyzing the characteristics of the various clusters. The algorithms implemented in CLUTO follow either the ***partitional*** or the ***agglomerative*** paradigm for cluster discovery [1] and treat the clustering problem as an optimization process which seeks to maximize or minimize a particular ***clustering criterion function*** defined over the entire clustering solution. CLUTO provides a total of seven different criterion functions that are described and analyzed in [2]. Most of these criterion functions have been shown to produce high quality clustering solutions in high dimensional datasets, especially those arising in document clustering.

An important aspect of partitional-based criterion-driven clustering algorithms is the method used to optimize this criterion function. CLUTO uses a randomized incremental optimization algorithm that is greedy in nature, has low computational requirements, and has been shown to produce high-quality clustering solutions [2].

In addition to the clustering algorithms, CLUTO provides tools for analyzing the discovered clusters to understand the relations between the objects assigned to each cluster and the relations between the different clusters, and tools for visualizing the discovered clustering solutions. In particular, CLUTO can identify the features that best describe and/or discriminate each cluster. These set of features can be used to gain a better understanding of the set of objects assigned to each cluster and to provide concise summaries about the cluster's contents. Moreover, CLUTO provides visualization capabilities that can be used to see the relationships between the clusters, objects, and features.

CLUTO's distribution consists of both stand-alone programs (vpcluster and vacluster) for clustering and analyzing these clusters, as well as a library via which an application program can access directly the various clustering and analysis algorithms implemented in CLUTO.

1.2 When CLUTO Should be Used

CLUTO's primary clustering algorithms treat the objects to be clustered as vectors in a high-dimensional space and measure the degree of similarity between these objects using either the cosine function between these vectors or the Pearson's correlation coefficient. Using these measures, two objects are similar if their corresponding vectors [1] point in the same direction (*i.e.*, they have roughly the same set of features and in the same proportion), regardless of their actual length.

These cosine- and correlation-based similarity measures are well-suited for clustering high-dimensional (as well as low-dimensional) datasets arising in many diverse applications areas, including information retrieval, customer purchasing transactions, science, and biology. Moreover, for many criterion functions, clustering algorithms based on the cosine similarity measure are equivalent with algorithms that use the Euclidean distance measure on vectors that are scaled to be of unit-length [2].

CLUTO's algorithms have been optimized for operating on very large datasets both in terms of the number of objects as well as the number of dimensions. This is especially true for CLUTO's algorithms for partitional clustering. These algorithms can quickly cluster datasets with several tens of thousands objects and several thousands of dimensions. Moreover, since most high-dimensional datasets are very sparse, CLUTO directly takes into account this sparsity and requires memory that is roughly linear on the input size.

[1] In the case of Pearson's correlation coefficient the vectors are obtained by first subtracting their average value.

1.3 Outline of CLUTO's Manual

CLUTO's manual is organized as follows. Section 2 describe the major changes from the previous release. Section 3 describes the stand-alone program provided by CLUTO, and discusses its various options and analysis capabilities. Section 4 describes the application programming interface (API) of the stand-alone library that implements the various algorithms implemented in CLUTO. Finally, Section 5 describes the system requirements for the CLUTO package.

2 Major Changes From Release 1.0

The latest release of CLUTO contains five major additions over its earlier release, and a number of minor changes. The major changes are the following:

1. CLUTO now includes a fully-functional set of agglomerative clustering routines and an agglomerative stand-alone program called vacluster. CLUTO's agglomerative routines compute an agglomerative clustering solution that locally optimizes any one of its various clustering criterion functions. These routines are reasonably efficient and they can cluster datasets having up to 3,000–6,000 objects. Of course, due to their agglomerative nature, their space and memory complexity is at least quadratic on the number of objects. For this reason, the partitional clustering algorithms should be preferred for large datasets.

2. CLUTO's agglomerative routines can also be used to cluster a set of objects whose relations are defined in terms of an arbitrary object-to-object similarity matrix. This functionality is currently available only via the use of the CLUTO_SA_Cluster() routine in CLUTO's library.

3. CLUTO can now cluster objects using the correlation coefficient between their vectors as a measure of their similarity. This eliminates the need for pre-processing.

4. CLUTO's stand-alone programs and library now support dense datasets. This allows CLUTO to be used directly to cluster datasets arising in scientific data and microarray expression analysis.

5. CLUTO can now produce a number of different graphical visualizations of the discovered clusters. These visualizations can be used to see the relationships between the various clusters, objects, and features. Moreover, CLUTO supports many widely used output formats including postscript, Adobe Illustrator, XFig, XML-based SVG, PCL, WebCGM, and GIF.

3 Using CLUTO via its Stand-Alone Program

CLUTO provides access to its various clustering and analysis algorithms via the vpcluster and vacluster stand-alone cluster programs. The rest of this section describes how to use these programs, how to interpret their output, the format of the various input files that they require, and the format of the output files that they produce.

3.1 The vpcluster Clustering Program

The vpcluster program is used to cluster a collection of objects into a predetermined number of clusters k. The vpcluster program treats each object as a vector in a high-dimensional space, and it computes the clustering solution using the ***partitioning*** paradigm, whose goal is to optimize (minimize or maximize) a particular function that measures the quality of the clustering solution. This vector-based representation and partitional-principle of vpcluster's is the reason for the "vp" prefix of vpcluster's name.

The vpcluster program is invoked by providing two required parameters on the command line along with a number of optional parameters. Its overall calling sequence is as follows:

vpcluster [optional parameters] *MatrixFile* *NClusters*

The first required argument, *MatrixFile*, is the name of the file that stores the n objects that need to be clustered. In vpcluster, each one of these objects is considered to be a vector in an m-dimensional space. The collection of these objects is treated as an $n \times m$ matrix, whose rows correspond to the objects, and whose columns correspond to the dimensions of the feature space. The exact format of the matrix-file is described in Section 3.4.1. The second required argument *NClusters*, is the number of clusters that is desired.

Upon successful execution, vpcluster displays statistics regarding the quality of the computed clustering solution and the amount of time taken to perform the clustering. The actual clustering solution is stored in a file named ***MatrixFile.clustering.NClusters***, whose format is described in Section 3.5.1.

Figure 1 shows the output of vpcluster for clustering a matrix into 10 clusters. From this figure we see that vpcluster initially prints information about the matrix, such as its name, the number of rows (*#Rows*), the number of columns (*#Columns*), and the number of non-zeros in the matrix (*#NonZeros*). Next it prints information about the values of the various options that it used to compute the clustering (we will discuss the various options in Section 3.2.1), and the number of desired clusters (*#Clusters*). Once it computes the clustering solution, it displays information regarding the quality of the overall clustering solution as well as the quality of each cluster. The meaning of the various measures that are reported will be discussed in Section 3.3. Finally, vpcluster reports the time taken by the various phases of the program. For this particular example, vpcluster required 0.950 seconds to read the input file and write the clustering solution, 10.220 seconds to compute the actual clustering solution, and 0.220 seconds to compute statistics on the quality of the clustering.

3.1.1 Optional Parameters

The behavior of vpcluster can be controlled by specifying a number of different optional parameters. These parameters can be broadly categorized into two groups. The first group controls various aspects of the clustering algorithm, whereas the second group controls the type of reporting and analysis that vpcluster performs on the computed clusters.

The optional parameters are specified using the standard `-paramname` or `-paramname=value` formats, where the name of the optional parameter `paramname` can be truncated to a unique prefix of the parameter name.

Clustering Algorithm Parameters There are a total of 13 different optional parameters that control how vpcluster computes the clustering solution. The name and function of these parameters is as follows:

-denseinput

This parameter instructs vpcluster to use the dense matrix storage format while reading the input *MatrixFile* matrix. If this parameter is omitted, then vpcluster assumes that the matrix is stored using a sparse format.

```
prompt% vpcluster sports.mat 10
*******************************************************************************
 CLUTO 1.5 - vpcluster, Copyright 2001-02, Regents of the University of Minnesota

Matrix Information ----------------------------------------------------------
  Name: sports.mat, #Rows: 8580, #Columns: 126373, #NonZeros: 1107980

Options ---------------------------------------------------------------------
  CRfun=I2, RowModel=None, ColModel=IDF, Prune=1.00, NTrials=10, NIter=10
  Direct=NO, RB=YES, Refine=NO, #Clusters: 10

Solution --------------------------------------------------------------------

------------------------------------------------------------------------
10-way clustering solution: [I2=2.28e+03]
------------------------------------------------------------------------
cid  Size  ISim ISdev  ESim ESdev |
------------------------------------------------------------------------
  0   795 0.102 0.036 0.018 0.006 |
  1   628 0.106 0.041 0.022 0.007 |
  2   760 0.099 0.034 0.021 0.006 |
  3   845 0.087 0.034 0.020 0.007 |
  4   845 0.095 0.035 0.023 0.007 |
  5   848 0.104 0.038 0.025 0.009 |
  6  1168 0.051 0.016 0.021 0.006 |
  7   831 0.045 0.015 0.019 0.005 |
  8   984 0.032 0.012 0.015 0.006 |
  9   876 0.046 0.016 0.023 0.008 |
------------------------------------------------------------------------

Timing Information
-------------------------------------------------------------
   I/O:                                      0.950 sec
   Clustering:                              10.220 sec
   Reporting:                                0.240 sec
*******************************************************************************
```

Figure 1: Output of vpcluster for matrix *sports.mat* and a 10-way clustering.

-rb This parameter instructs vpcluster to compute the desired k-way clustering solution by performing a sequence of $k-1$ ***repeated bisections***. In this approach, the matrix is first clustered into two groups, then one of these groups is selected and bisected further. This process continuous until the desired number of clusters is found. During each step, vpcluster selects the largest cluster to bisect further. Note that this approach ensures that the criterion function is locally optimized within each bisection, but in general is not optimized globally. By default, vpcluster uses this approach to find the k-way clustering solution.

-direct This parameter instructs vpcluster to compute the desired k-way clustering solution, by simultaneously finding all k clusters. In general, computing a k-way clustering directly is slower than clustering via repeated bisections. In terms of quality, for reasonably small values of k (usually less than 20), the direct approach leads to better clusters than those obtained via repeated bisections. However, as k increases, the *-rb* approach tends to be better than *-direct*. Please refer to [2] (which is included with CLUTO' distribution) for a detailed comparisons of the *-direct* and *-rb* approach in the context of clustering document datasets.

-refine This parameter instructs vpcluster to globally optimize the clustering solution produced by the repeated bisecting approach. Essentially, vpcluster uses the solution obtained by *-rb* as the initial clustering solution and tries to optimize it further using its optimizer.

-fulltree Builds a complete hierarchical tree that preserves the clustering solution that was computed. In this hierarchical clustering solution, the objects of each cluster form a subtree, and the different subtrees are merged to get an all inclusive cluster at the end. The hierarchical agglomerative clustering is computed so that it optimizes the selected clustering criterion function (specified by *-crfun*). If *-writetree* is specified, the resulting hierarchical agglomerative tree is stored in the appropriate file. This option should be used to obtain a hierarchical agglomerative clustering solution for very large data sets, and for re-ordering the rows of the matrix when *-plotmatrix* is specified.

-sim=int Selects the similarity function to be used for clustering. The possible values are:

1 The similarity between objects is computed using the cosine function. This is the default setting.

2 The similarity between objects is computed using the correlation coefficient.

The runtime of vpcluster may increase when *-sim=2* is selected, as it needs to store and operate on the dense $n \times m$ matrix.

-crfun=int

This parameter selects the particular clustering criterion function to be used in finding the clusters. A total of seven different clustering criterion functions are provided that are selected by specifying the appropriate integer value. The possible values for *-crfun* are:

1 Selects the I1 criterion function ($\mathcal{I}_1$ in [2]).

2 Selects the I2 criterion function ($\mathcal{I}_2$ in [2]). This is the default setting.

3 Selects the E1 criterion function ($\mathcal{E}_1$ in [2]).

4 Selects the G1 criterion function ($\mathcal{G}_1$ in [2]).

5 Selects the G1' criterion function ($\mathcal{G}'_1$ in [2]).

6 Selects the H1 criterion function ($\mathcal{H}_1$ in [2]).

7 Selects the H2 criterion function ($\mathcal{H}_2$ in [2]).

The precise mathematical definition of these criterion functions is beyond the scope of this manual, and the reader is referred to [2] for both a detailed description and evaluation of the various criterion functions.

The various criterion functions can sometimes lead to significantly different clustering solutions. In general, the $\mathcal{I}_2$ and $\mathcal{H}_2$ criterion functions lead to very good clustering solutions, whereas the $\mathcal{E}_1$ and $\mathcal{G}'_1$ criterion functions leads to solutions that contain clusters that are of comparable size. However, the choice of the *right* criterion function depends on the underlying application area, and the user should perform some experimentation before selecting one appropriate for his/her needs.

-rowmodel=int

Selects the model to be used to scale the various columns of each row. The possible values are:

1 The columns of each row are not scaled and used as they are provided in the input file. This is the default setting.

2 The columns of each row are scaled so that their values are between 0.5 and 1.0. In particular, the jth column of the ith row of the matrix ($r_{i,j}$) is scaled to be equal to

$$r'_{i,j} = 0.5 + 0.5\frac{r_{i,j}}{max_l(r_{i,l})}.$$

This scaling was motivated by a similar scaling of document vectors in information retrieval, and it is referred to as the *MAXTF* scaling scheme.

3 The columns of each row are scaled to be equal to the square-root of their actual values. That is, $r'_{i,j} = \text{sign}(r_{i,j})\sqrt{r_{i,j}}$, where $\text{sign}(r_{i,j})$ is 1.0 or -1.0, depending on whether or not $r_{i,j}$ is positive or negative. This scaling is referred to as the *SQRT* scaling scheme.

4 The columns of each row are scaled to be equal to the log of their actual values. That is, $r'_{i,j} = \text{sign}(r_{i,j}) \log_2 r_{i,j}$. This scaling is referred to as the *LOG* scaling scheme.

The last three scaling schemes are primarily used to smooth large values in certain columns (*i.e.*, dimensions) of each vector.

-colmodel=int

Selects the model to be used to scale the various columns globally across all the rows. The possible values are:

1 The columns of the matrix are not globally scaled, and they are used as is. This is the default setting used by vpcluster when the correlation coefficient-based similarity function is used.

2 The columns of the matrix are scaled according to the *inverse-document-frequency* (IDF) paradigm, used in information retrieval. In particular, if rf_i is the number of rows that the ith column belongs to, then each entry of the ith column is scaled by $-\log_2(rf_i/n)$. The effect of this scaling is to de-emphasize columns that appear in many rows. This is the default setting used by vpcluster when the cosine similarity function is used.

The global scaling of the columns occurs after the per-row column scaling selected by the *-rowmodel* parameter has been performed.

-prune=float

Selects the factor by which vpcluster will prune the columns before performing the clustering. This is a number p between 0.0 and 1.0 and indicates the fraction of the overall similarity that the retained columns must account for. For example, if $p = 0.9$, vpcluster first determines how much each column contributes to the overall pairwise similarity between the rows, and then selects as many of the highest contributing columns as required to account for 90% of the similarity. Reasonable values are within the range of $(0.8 \cdots 1.0)$, and the default value used by vpcluster is 1.0, indicating that no columns will be pruned. In general, this parameter leads to a substantial reduction of the number of columns (*i.e.*, dimensions) without seriously affecting the overall clustering quality.

-ntrials=int

Selects the number of different clustering solutions to be computed. If l is the supplied number, then vpcluster computes a total of l clustering solutions (each one of them starting with a different set of seed objects), and then selects the solution that has the best value of the criterion function that was used. The default value for vpcluster is 10.

-niter=int

Selects the maximum number of refinement iterations to be performed, within each clustering step. Reasonable values for this parameter are usually in the range of 5–20. The default value of vpcluster is 10.

-seed=int Selects the seed of the random number generator to be used in vpcluster.

Reporting and Analysis Parameters There are a total of 13 different optional parameters that control the amount of information that vpcluster reports about the clusters as well as the analysis that it performs on the discovered clusters. The name and function of these parameters is as follows:

-nooutput

Specifies that vpcluster should not write the clustering vector onto the disk.

-output=string

Specifies the name of the file onto which the clustering vector should be written. If this parameter is not specified, then the clustering vector is written to the ***MatrixFile.clustering.NClusters*** file, where *MatrixFile* is the name of the file that stores the matrix to be clustered, and *NClusters* is the number of desired clusters.

-clabelfile=string

Specifies the name of the file that stores the labels of the columns. The labels of the columns are used for reporting purposes when the *-showfeatures* or the *-labeltree* options are specified. The format of this file is described in Section 3.4.3. If this parameter is not specified, vpcluster looks to see if a file called ***MatrixFile.clabel*** exists, and if it does, reads this file, instead. If no file is provided or the default file does not exist, then the label of the jth column becomes "colj" (*i.e.*, it is labeled by its corresponding column-id).

-rlabelfile=string

Specifies the name of the file that stores the labels of the rows. The labels of the rows are used for reporting purposes when the *-plotmatrix* or the *-plotclusters* options are specified. The format of this file is described

in Section 3.4.2. If this parameter is not specified, vpcluster looks to see if a file called ***MatrixFile.rlabel*** exists, and if it does, reads this file, instead. If no file is provided or the default file does not exist, then the label of the jth row becomes "rowj" (*i.e.*, it is labeled by its corresponding row-id).

-rclassfile=string

Specifies the name of the file that stores the class-labels of the rows (*i.e.*, the objects to be clustered). This is used by vpcluster to compute the quality of the clustering solution using external quality measures and to output how the objects of different classes are distributed among clusters. The format of this file is described in Section 3.4.4. If this parameter is not specified, vpcluster looks to see if a file called ***MatrixFile.rlabel*** exists, and if it does, reads this file, instead. If no file is provided or the default file does not exist, vpcluster assumes that the class labels of the objects are not known and does not perform any cluster-quality analysis based on external measures.

-showfeatures

This parameter instructs vpcluster to analyze the discovered clusters and identify the set of features (*i.e.*, columns of the matrix) that are most descriptive of each cluster, as well as the set of features that best discriminate each cluster from the rest of the objects. The set of *descriptive* features is determined by selecting the columns that contribute the most to the average similarity between the objects of each cluster. On the other hand, the set of *discriminating* features is determined by selecting the columns that are more prevalent in the cluster compared to the rest of the objects. In general, there will be a large overlap between the descriptive and discriminating features. However, in some cases there may be certain differences, especially when *-colmodel=1*.

-nfeatures=int

Specifies the number of descriptive and discriminating features to display for each cluster when the *-showfeatures* or *-labeltree* options are used. The default value for this parameter is five (5).

-showtree=[int]

This parameter instructs vpcluster to build a hierarchical agglomerative tree on top of the clustering solution that was obtained. This tree will have *NClusters* leaves, each one corresponding to one of the discovered clusters, and provides a way of visualizing how the different clusters are related to each other. It takes as a parameter the clustering criterion function that is optimized in the process of building the tree. That is, the pairs of clusters that are selected and merged in each step are selected in such a fashion so that the resulting solution will optimize the specified clustering criterion function. The values and meaning of this parameter are identical to those used by *-crfun*. If no value is specified, then the tree is built using the same criterion function as that used in finding the clusters.

Note that the tree built by *-showtree* is a subset of the tree built by *-fulltree*, as the later also performs a hierarchical clustering within each cluster.

-labeltree

This parameter instructs vpcluster to label the nodes of the tree with the set of features that best describe the corresponding clusters. The method used for determining these features is identical to that used in *-showfeatures*. Note that the descriptive features for both the leaves (*i.e.*, original clusters) as well as the internal nodes of the tree are displayed. The number of features that is displayed is controlled by the *-nfeatures* parameter.

-writetree=[string]

This parameter instructs vpcluster to write the hierarchical agglomerative tree into a file. The format of this file is described in Section 3.5.2. The tree being written to the file can be either the tree built on top of the clustering solution (*i.e.*, the one displayed by *-showtree*), or the clustering-preserving tree produced when *-fulltree* was specified. This parameter takes an optional argument which is the name of the output file. If no name is specified, the tree is written to the *MatrixFile.tree.NClusters* file, or if *-fulltree* has also

been specified it is written to the *MatrixFile.tree.Nrows* file, where 'Nrows' is the number of rows in the matrix.

-zscores This parameter instructs vpcluster to analyze each cluster and for each object to output the z-score of its similarity to the other objects in its own cluster (internal z-score) as well as the objects of the different clusters (external z-score). The various z-score values are stored in the clustering file whose format is described in Section 3.5.1.

The internal z-score of an object j that is part of the lth cluster is given by $(s_j^I - \mu_l^I)/\sigma_l^I$, where s_j^I is the average similarity between the jth object and the rest of the objects in its cluster, μ_l^I is the average of the various s_j^I values over all the objects in the lth, and σ_l^I is the standard deviation of these similarities.

The external z-score of an object j that is part of the lth cluster is given by $(s_j^E - \mu_l^E)/\sigma_l^E$, where s_j^E is the average similarity between the jth object and the objects in the other clusters, μ_l^E is the average of the various s_j^E values over all the objects in the lth cluster, and σ_l^E is the standard deviation of these similarities.

Objects that have large values of the internal z-score and small values of the external z-score will tend to form the *core* of their clusters.

-dbglvl=int

Selects the level of debugging information to be printed. The value is obtained by adding the codes for the various options. The possible values for this option are:

1 Prints information about the clustering process.

2 Prints information about the criterion optimization process.

4 Prints information about the tree-building phase.

The default value of this parameter is zero, indicating that no debugging information is printed.

-help This options instructs vpcluster to print a short description of the various command line parameters.

Cluster Visualization Parameters The vpcluster clustering program can also produce visualizations of the computed clustering solutions. These visualizations are relatively simple plots of the original input matrix that show how the different objects (*i.e.*, rows) and features (*i.e.*, columns) are clustered together. There are a total of six optional parameters that control the type of visualization that vpcluster performs. The name and function of these parameters is as follows:

-plotformat=string

Selects the format of the graphics files produced by vpcluster's visualization options. The possible values for this option are:

ps Outputs an encapsulated postscript[2] file. This is the default option.

fig Outputs the visualization in a format that is compatible with the Unix XFig program. This file can then be edited with XFig.

ai Outputs the visualization in a format that is compatible with the Adobe Illustrator program. This file can then be edited with Illustrator or other programs that understand this format (*e.g.*, Visio).

svg Outputs the visualization in the XML-based Scalable Vector Format that can be viewed by modern web-browsers (if the appropriate plug-in is installed).

[2] Sometimes, while trying to convert the postscript files generated by CLUTO into PDF format using Adobe's distiller you may notice that the text is not included in the PDF file. To correct this problem reconfigure your distiller not to include *truetype* fonts when the required text font is part of the standard postscript fonts.

cgm Outputs the visualization in the WebCGM format.

pcl Outputs the visualization in HP's PCL 5 format used by many laserjet or compatible printers.

gif Outputs the visualization in widely used GIF bitmap format.

-plotmatrix=string

Instructs vpcluster to produce a visualization that shows how the rows of the original matrix are clustered together. This is done by showing an appropriate row- and possibly a column-permutation of the original matrix, along with a color-intensity plot of the various values of the matrix. The actual visualization is stored in the file whose name is supplied as an option to *-plotmatrix*.

In this matrix permutation, the rows of the matrix assigned to the same cluster are re-ordered to be at consecutive rows, followed by a reordering of the clusters. The actual ordering of the rows and clusters depends on whether the *-fulltree* parameter was specified. If it was not specified, then the clusters are ordered according to their cluster-id number, and within each cluster the rows are numbered according to the row-id number. However, if *-fulltree* was specified, both the rows and the clusters are re-ordered according the hierarchical tree computed by *-fulltree*. In addition to that, the actual tree is drawn along the side of the matrix.

If the input matrix is in dense format, then *-plotmatrix* displays the columns, in column-id order. If the *-clustercolumns* option was specified, then the columns are re-ordered according to a hierarchical clustering solution of the columns.

If the matrix is sparse, only a subset of the columns is displayed, that corresponds to the union of the descriptive and discriminating features of each cluster computed by *-showfeatures*. The number of features from each cluster that is included in that union can be controlled by the *-nfeatures* parameter. Again, the columns can be displayed in either the column-id order or if the *-clustercolumns* option was specified, then the columns are re-ordered according to a hierarchical clustering solution of the columns.

The labels printed along each row and column of the matrix can be specified by using the *-rlabelfile* and *-clabelfile*, respectively.

The plot uses red to denote positive values and green to denote negative values. Bright red/green indicate large positive/negative values, whereas colors close to white indicate values close to zero.

-plotclusters=string

Instructs vpcluster to produce a visualization that shows how the clusters are clustered together, by showing a color-intensity plot of the various values in the various cluster centroid vectors. The actual visualization is stored in the file whose name is supplied as an option to *-plotclusters*.

The produced visualization is similar to that produced by *-plotmatrix*, but now only *NClusters* rows are shown, one for each cluster. The height of each row is proportional to the log of the corresponding cluster's size. The ordering of the clusters is determined by computing a hierarchical clustering (similar to that produced via *-showtree*), and the ordering of the columns is controlled by the *-clustercolumns* parameter.

The column selection mechanism and color-scheme are identical to that used by *-plotmatrix*.

-clustercolumns

Instructs vpcluster to compute a hierarchical clustering of the columns and to reorder them when *-plotmatrix* and *-plotclusters* is specified. This can be used to generate a visualization in which the features are clustered together.

-noreorder

Instructs vpcluster not to try to produce a visually pleasing reordering of the various hierarchical trees that is drawing. This option is turned off by default if the number of objects that are clustered is greater than 4000.

-zeroblack

Instructs vpcluster to use black color for denoting zero (or small values) in the matrix.

3.2 The vacluster Clustering Program

The vacluster program is used to cluster a collection of objects into a predetermined number of clusters k. The vacluster program, treats each object as a vector in a high-dimensional space, and it computes the clustering solution using the ***agglomerative*** paradigm whose goal is to locally optimize (minimize or maximize) a particular function that measures the quality of the clustering solution. This vector-based representation and agglomerative-principle of vacluster's is the reason for the "va" prefix of vacluster's name. Thus, the key difference between vacluster and vpcluster is that the first uses an agglomerative approach to locally optimize the clustering criterion function, whereas the later uses a partitional approach to either globally or locally optimize the clustering criterion function.

The vacluster program is invoked by providing two required parameters on the command line along with a number of optional parameters–similar to the way vpcluster is invoked. Its overall calling sequence is as follows:

vacluster [optional parameters] *MatrixFile* *NClusters*

As was the case with vpcluster, the first required argument *MatrixFile*, is the name of the file that stores the n objects that need to be clustered, and the second required argument *NClusters*, is the number of clusters that is desired. Upon successful execution, vacluster displays statistics regarding the quality of the computed clustering solution and the amount of time taken to perform the clustering. The actual clustering solution is stored in a file named ***MatrixFile.clustering.NClusters***, whereas the complete hierarchical agglomerative tress is stored in a file name ***MatrixFile.tree.NRows*** (where *"Nrows"* is the number of the rows of the input matrix. The format of these files are described in Section 3.5.1 and 3.5.2.

Because vacluster's clustering algorithms are agglomerative in nature, they have a much higher computational and memory complexity than the partitional algorithms used by vpcluster. For this reason, vacluster can only be used to cluster datasets that have fewer than 3000–6000 objects, depending on the available memory and the particular criterion function that was selected.

Figure 2 shows the output of vacluster for clustering a matrix into 10 clusters. From this figure we see that vacluster's output is similar to that produced by vpcluster (even though there are some slight differences),

```
prompt% vacluster k1b.mat 10
*******************************************************************************
 CLUTO 1.5 - vacluster, Copyright 2001-02, Regents of the University of Minnesota

Matrix Information ----------------------------------------------------------
  Name: k1b.mat, #Rows: 2340, #Columns: 21839, #NonZeros: 349792

Options ---------------------------------------------------------------------
  CRfun=I2, RowModel=None, ColModel=IDF, Prune=1.00, SimFun=Cosine, #Clusters: 10

Solution --------------------------------------------------------------------

------------------------------------------------------------------------
10-way clustering solution: [I2=5.28e+02]
------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  |
------------------------------------------------------------------------
  0   472 +0.047 +0.013 +0.010 +0.003 |
  1   142 +0.068 +0.021 +0.010 +0.002 |
  2   624 +0.027 +0.008 +0.015 +0.005 |
  3    77 +0.156 +0.041 +0.017 +0.005 |
  4   111 +0.107 +0.055 +0.015 +0.003 |
  5   399 +0.028 +0.010 +0.015 +0.004 |
  6    67 +0.232 +0.077 +0.022 +0.008 |
  7   155 +0.055 +0.022 +0.018 +0.005 |
  8   157 +0.078 +0.038 +0.018 +0.007 |
  9   136 +0.084 +0.038 +0.015 +0.004 |
------------------------------------------------------------------------

Timing Information ----------------------------------------------------------
   I/O:                                    0.930 sec
   Clustering:                             7.660 sec
   Reporting:                              0.290 sec
*******************************************************************************
```

Figure 2: Output of vacluster for matrix *k1b.mat* and a 10-way clustering.

3.2.1 Optional Parameters

The behavior of vacluster can be controlled by specifying a number of different optional parameters. Because many of vacluster's parameters as similar to those used by vpcluster, our description will only focus on describing the differences between them.

Clustering Algorithm Parameters There are a total of six different optional parameters that control how vacluster computes the clustering solution. The name and function of these parameters is as follows:

-denseinput
Identical to vpcluster's corresponding parameter.

-sim=int Identical to vpcluster's corresponding parameter.

-crfun=int
Identical to vpcluster's corresponding parameter.

Note that the computational complexity of the vacluster program depends on the criterion function that is selected. In particular, if n is the number of objects, the complexity for the $\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{E}_1$, $\mathcal{G}_1$, and $\mathcal{G}'_1$ is $O(n^2 \log n)$, whereas vacluster's complexity for the $\mathcal{H}_1$ and $\mathcal{H}_2$ criterion functions is $O(n^4)$. The very high complexity for $\mathcal{H}_1$ and $\mathcal{H}_2$ is due to the fact that these two criterion functions are defined globally over the entire solution and they cannot be accurately evaluated based on the local combination of two clusters.

-rowmodel=int
Identical to vpcluster's corresponding parameter.

-colmodel=int
Identical to vpcluster's corresponding parameter.

-prune=float
Identical to vpcluster's corresponding parameter.

Reporting and Analysis Parameters There are a total of 13 different optional parameters that control the amount of information that vacluster reports about the clusters as well as the analysis that it performs on the discovered clusters. The name and function of these parameters is as follows:

-nooutput
Identical to vpcluster's corresponding parameter.

-clustfile=string
Identical to vpcluster's *-output* parameter.

-treefile=string
Specifies the name of the file into which the agglomerative tree will be written. The format of this tree is described in Section 3.5.2. If this parameter is not specified, then the tree is written to the ***MatrixFile.tree.NRows*** file, where *MatrixFile* is the name of the file that stores the matrix to be clustered, and *NRows* is the number of rows in the matrix file.

-clabelfile=string
Identical to vpcluster's corresponding parameter.

-rlabelfile=string
Identical to vpcluster's corresponding parameter.

-rclassfile=string
Identical to vpcluster's corresponding parameter.

-showfeatures
Identical to vpcluster's corresponding parameter.

-nfeatures=int
Identical to vpcluster's corresponding parameter.

-showtree=[int]
Identical to vpcluster's corresponding parameter.

-labeltree
Identical to vpcluster's corresponding parameter.

-zscores Identical to vpcluster's corresponding parameter.

-dbglvl=int
Selects the level of debugging information to be printed. The value is obtained by adding the codes for the various options. The possible values for this option are:

1 Prints progress information about the object-to-object similarity computational phase.

2 Prints information about the agglomeration progress.

The default value of this parameter is zero, indicating that no debugging information is printed.

-help Identical to vpcluster's corresponding parameter.

Cluster Visualization Parameters The vacluster clustering program can also produce visualizations of the computed clustering solutions, similar to those produced by vpcluster. There are a total of seven optional parameters that control the type of visualization that vacluster performs. The name and function of these parameters is as follows:

-plotformat=string
Identical to vpcluster's corresponding parameter.

-plottree=string
This parameter instructs vacluster to produce a graphic representation of the entire hierarchical tree produced by vacluster. The leaves of this tree are labeled based on the supplied row labels (*i.e.*, via the *-rlabelfile* parameter).

-plotmatrix=string
This parameter is identical to vpcluster's corresponding parameter with the only difference being that both the rows of the matrix as well as the clusters are automatically re-ordered according to the computed hierarchical agglomerative tree.

-plotclusters=string
Identical to vpcluster's corresponding parameter.

-clustercolumns
Identical to vpcluster's corresponding parameter.

-noreorder
Identical to vpcluster's corresponding parameter.

-zeroblack
Identical to vpcluster's corresponding parameter.

3.3 Understanding the Information Produced by vpcluster and vacluster

From the description of vpcluster's and vacluster's parameters we can see that they can output a wide-range of information and statistics about the clusters that they find. In the rest of this section we describe the format and meaning of these statistics. Most of our discussion will focus on vpcluster's output, since it is similar to that produced by vacluster.

3.3.1 Internal Cluster Quality Statistics

The simpler statistics reported by vpcluster & vacluster have to do with the quality of each cluster as measured by the criterion function that it uses and the similarity between the objects in each cluster. In particular, as the example in Figure 1 shows, the "*Solution*" section of vpcluster's output displays information about the clustering solution.

The first statistic that it reports is the overall value of the criterion function for the computed clustering solution. In our example, this is reported as "`I2=2.28e+03`", which is the value of the $\mathcal{I}_2$ criterion function of the resulting solution. If a different criterion function is specified (by using the *-crfun* option), then the overall cluster quality information will be displayed with respect to that criterion function.

After that, vpcluster then displays a table in which each row contains various statistics for each one of the clusters. The meaning of the columns of this table is as follows. The column labeled "cid" corresponds to the cluster number (or cluster id). The column labeled "Size" displays the number of objects that belongs to each cluster. The column labeled "ISim" displays the average similarity between the objects of each cluster (*i.e.*, internal similarities). The column labeled "ISdev" displays the standard deviation of these average internal similarities (*i.e.*, internal standard deviations). The column labeled "ESim" displays the average similarity of the objects of each cluster and the rest of the objects (*i.e.*, external similarities). Finally, the column labeled "ESdev" display the standard deviation of the external similarities (*i.e.*, external standard deviations).

Note that the clusters discovered by vpcluster are ordered in increasing (ISIM-ESIM) order. In other words, clusters that are *tight* and far away from the rest of the objects have smaller cid values.

3.3.2 External Cluster Quality Statistics

In addition to the internal cluster quality measures, vpcluster & vacluster can also take into account information about the classes that the various objects belong to (via the *-rclassfile* option) and compute various statistics that determine the quality of the clusters using that information. These statistics are usually referred to as *external quality measures* as the quality is determined by looking at information that was not used while finding the clustering solution.

Figure 3 shows the output of vpcluster when such a class file is provided for our example *sports.mat* dataset. This dataset contains various documents that talk about seven different sports (baseball, basketball, football, hockey, boxing, bicycling, and golfing), and each document (*i.e.*, object to be clustered) belongs to one of these topics. Once vpcluster finds the 10-way clustering solution, it then uses this class information to analyze both the quality of the overall clustering solution as well as the quality of each cluster. Also note that vpcluster was invoked with the *-refine* option to performs a global refinement on the clustering solution.

Looking at Figure 3 we can see that vpcluster, in addition to the overall value of the criterion function, now prints the ***entropy*** and the ***purity*** of the clustering solution. For the exact formula of how the entropy and purity of the clustering solution is computed, please refer to [2]. Small entropy values and large purity values indicate good clustering solutions.

In addition to these measures, the cluster information table now contains two additional sets of information. The first set is the entropy and purity of each cluster and is displayed in the columns labeled "Entpy" and "Purty", respectively. The second set is information about how the different classes are distributed in each one of the clusters. This information is displayed in the last seven columns of this table, whose column labels are derived from the first four characters if the class names. That is "base" corresponds to baseball, "bask" corresponds to basketball, and so on. Each column shows the number of documents of this class that are in each cluster. For example, the first cluster contains one document about baseball, one document about basketball, and 786 documents about hockey. Looking at this class-distribution table, we can easily determine the quality of the different clusters.

Finally, comparing this clustering solution with the one displayed in Figure 1 we can see that by performing a final k-way refinement, the value of the criterion function improves. In this case, $\mathcal{I}_2$'s criterion function is now $2.30e+03$ which is greater than the earlier value of $2.28e+03$. Note that the goal of the clustering algorithm with respect to the $\mathcal{I}_2$ criterion function is to maximize its value.

```
prompt% vpcluster -refine -rclassfile=sports.rclass sports.mat 10
*******************************************************************************
 CLUTO 1.5 - vpcluster, Copyright 2001-02, Regents of the University of Minnesota

Matrix Information ------------------------------------------------------------
  Name: sports.mat, #Rows: 8580, #Columns: 126373, #NonZeros: 1107980

Options -----------------------------------------------------------------------
  CRfun=I2, RowModel=None, ColModel=IDF, Prune=1.00, NTrials=10, NIter=10
  Direct=NO, RB=YES, Refine=YES, #Clusters: 10

Solution ----------------------------------------------------------------------

-------------------------------------------------------------------------------
10-way clustering solution: [I2=2.30e+03], Entropy: 0.169, Purity: 0.878
-------------------------------------------------------------------------------
cid  Size  ISim ISdev  ESim ESdev Entpy Purty | base bask foot hock boxi bicy golf
-------------------------------------------------------------------------------
  0   788 0.103 0.036 0.018 0.006 0.010 0.997 |    1    1    0  786    0    0    0
  1   682 0.109 0.035 0.021 0.006 0.011 0.997 |    1    0  680    0    0    0    1
  2   610 0.110 0.041 0.022 0.007 0.018 0.995 |  607    1    2    0    0    0    0
  3   818 0.090 0.034 0.020 0.007 0.014 0.996 |    0  815    2    1    0    0    0
  4   795 0.102 0.034 0.023 0.007 0.016 0.995 |  791    0    4    0    0    0    0
  5   863 0.103 0.038 0.025 0.009 0.000 1.000 |  863    0    0    0    0    0    0
  6   791 0.045 0.018 0.015 0.006 0.821 0.411 |   76   33   92    5  120  140  325
  7  1133 0.050 0.016 0.021 0.007 0.030 0.991 |    7    2 1123    1    0    0    0
  8   988 0.044 0.014 0.019 0.005 0.486 0.533 |   54  527  390    8    0    1    8
  9  1112 0.036 0.013 0.021 0.008 0.210 0.910 | 1012   31   53    8    2    4    2
-------------------------------------------------------------------------------

Timing Information ------------------------------------------------------------
   I/O:                                  1.830 sec
   Clustering:                          14.540 sec
   Reporting:                            0.220 sec
*******************************************************************************
```

Figure 3: Output of vpcluster for matrix *sports.mat* and a 10-way clustering that uses external quality measures.

3.3.3 Looking at each Cluster's Features

By specifying the *-showfeatures* option, vpcluster & vacluster will analyze each one of the clusters and determine the set of features (*i.e.*, columns of the matrix) that best describe and discriminate each one of the clusters. Figure 4 shows the output produced by vpcluster when *-showfeatures* was specified and when a file was provided with the labels of each one of the columns (via the *-clabelfile* option).

Looking at this figure, we can see that the set of descriptive and discriminating features are displayed right after the table that provides statistics for the various clusters. For each cluster, vpcluster displays three lines of information. The first line contains some basic statistics for each cluster (*e.g.*, cid, Size, ISim, ESim), whose meaning is identical to those displayed in the earlier table. The second line contains the five most descriptive features, whereas the third line contains the five most discriminating features. The features in these lists are sorted in decreasing descriptive or discriminating order. The reason that five features are printed is because this is the default value for the *-nfeatures* parameter; fewer or more features can be displayed by setting this parameter appropriately.

Right next to each feature, vpcluster displays a number that in the case of the descriptive features is the percentage of the within cluster similarity that this particular feature can explain. For example, for the 0th cluster, the feature "goal" explains 9.5% of the average similarity between the objects of the 0th cluster. A similar quantity is displayed for each one of the discriminating features, and is the percentage of the dissimilarity between the cluster and the rest of the objects which this feature can explain. In general there is a large overlap between descriptive and discriminating features, with the only difference being that the percentages associated with the discriminating features are typically smaller than the corresponding percentages of the descriptive features. This is because some of the descriptive features of a cluster may also be present in a small fraction of the objects that do not belong to this cluster.

If no labels for the different columns are provided, vpcluster outputs the column number of each feature instead of its label. This is illustrated in Figure 5 for the same problem in which *-clabelfile* was not specified. Note that the columns are numbered from one.

3.3.4 Looking at the Hierarchical Agglomerative Tree

The vpcluster & vacluster programs can also produce a hierarchical agglomerative tree in which the discovered clusters form the leaf nodes of this tree. This is done by specifying the *-showtree* parameter. In constructing this tree, the algorithms repeatedly merge a particular pair of clusters, and the pair of clusters to be merged is selected so that

```
prompt% vpcluster -refine -rclassfile=sports.rclass -clabelfile=sports.clabel -showfeatures sports.mat 10
*******************************************************************************
 CLUTO 1.5 - vpcluster, Copyright 2001-02, Regents of the University of Minnesota

Matrix Information ------------------------------------------------------------
  Name: sports.mat, #Rows: 8580, #Columns: 126373, #NonZeros: 1107980

Options -----------------------------------------------------------------------
  CRfun=I2, RowModel=None, ColModel=IDF, Prune=1.00, NTrials=10, NIter=10
  Direct=NO, RB=YES, Refine=YES, #Clusters: 10

Solution ----------------------------------------------------------------------

-------------------------------------------------------------------------------
10-way clustering solution: [I2=2.30e+03], Entropy: 0.169, Purity: 0.878
-------------------------------------------------------------------------------
cid  Size  ISim ISdev  ESim ESdev Entpy Purty | base bask foot hock boxi bicy golf
-------------------------------------------------------------------------------
  0   788 0.103 0.036 0.018 0.006 0.010 0.997 |    1    1    0  786    0    0    0
  1   682 0.109 0.035 0.021 0.006 0.011 0.997 |    1    0  680    0    0    0    1
  2   610 0.110 0.041 0.022 0.007 0.018 0.995 |  607    1    2    0    0    0    0
  3   818 0.090 0.034 0.020 0.007 0.014 0.996 |    0  815    2    1    0    0    0
  4   795 0.102 0.034 0.023 0.007 0.016 0.995 |  791    0    4    0    0    0    0
  5   863 0.103 0.038 0.025 0.009 0.000 1.000 |  863    0    0    0    0    0    0
  6   791 0.045 0.018 0.015 0.006 0.821 0.411 |   76   33   92    5  120  140  325
  7  1133 0.050 0.016 0.021 0.007 0.030 0.991 |    7    2 1123    1    0    0    0
  8   988 0.044 0.014 0.019 0.005 0.486 0.533 |   54  527  390    8    0    1    8
  9  1112 0.036 0.013 0.021 0.008 0.210 0.910 | 1012   31   53    8    2    4    2
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
10-way clustering solution - Descriptive & Discriminating Features...
-------------------------------------------------------------------------------
Cluster   0, Size:   788, ISim: 0.103, ESim: 0.018
      Descriptive:  shark 22.5%, goal  9.5%, nhl  4.3%, period  3.5%, penguin  1.6%
   Discriminating:  shark 17.2%, goal  6.0%, nhl  3.3%, period  2.3%, penguin  1.2%

Cluster   1, Size:   682, ISim: 0.109, ESim: 0.021
      Descriptive:  yard 37.6%, pass  8.1%, touchdown  6.9%, td  2.8%, kick  2.1%
   Discriminating:  yard 28.6%, pass  5.5%, touchdown  5.2%, td  2.2%, kick  1.5%

Cluster   2, Size:   610, ISim: 0.110, ESim: 0.022
      Descriptive:  canseco  9.1%, henderson  7.6%, russa  6.4%, la  3.9%, mcgwire  3.3%
   Discriminating:  canseco  7.4%, henderson  5.9%, russa  5.4%, mcgwire  2.7%, la  2.7%

Cluster   3, Size:   818, ISim: 0.090, ESim: 0.020
      Descriptive:  warrior 15.4%, laker  4.3%, hardawai  2.6%, mullin  2.4%, nba  2.2%
   Discriminating:  warrior 12.7%, laker  3.6%, hardawai  2.2%, mullin  2.0%, nba  1.7%

Cluster   4, Size:   795, ISim: 0.102, ESim: 0.023
      Descriptive:  giant 20.6%, mitchell  5.0%, craig  3.4%, mcgee  2.4%, clark  2.1%
   Discriminating:  giant 15.1%, mitchell  4.3%, craig  2.4%, mcgee  2.1%, clark  1.6%

Cluster   5, Size:   863, ISim: 0.103, ESim: 0.025
      Descriptive:  in  9.5%, hit  8.3%, homer  4.0%, run  4.0%, sox  2.1%
   Discriminating:  in  6.3%, hit  5.0%, homer  2.5%, run  1.7%, sox  1.4%

Cluster   6, Size:   791, ISim: 0.045, ESim: 0.015
      Descriptive:  box 31.6%, golf  3.9%, hole  3.0%, round  2.4%, par  2.1%
   Discriminating:  box 23.5%, golf  3.4%, hole  2.5%, par  1.8%, round  1.4%

Cluster   7, Size:  1133, ISim: 0.050, ESim: 0.021
      Descriptive:  seifert  3.5%, montana  3.2%, raider  2.6%, nfl  2.6%, bowl  2.4%
   Discriminating:  seifert  3.8%, montana  3.4%, raider  2.6%, nfl  2.6%, super  2.2%

Cluster   8, Size:   988, ISim: 0.044, ESim: 0.019
      Descriptive:  santa  2.4%, confer  2.3%, cal  2.3%, school  1.9%, st  1.7%
   Discriminating:  santa  2.2%, cal  2.0%, confer  1.8%, school  1.6%, stanford  1.4%

Cluster   9, Size:  1112, ISim: 0.036, ESim: 0.021
      Descriptive:  basebal  3.3%, million  2.3%, pitcher  2.0%, brave  1.9%, leagu  1.6%
   Discriminating:  basebal  3.9%, million  2.3%, brave  1.8%, pitcher  1.6%, outfield  1.2%
-------------------------------------------------------------------------------

Timing Information ------------------------------------------------------------
   I/O:                          1.530 sec
   Clustering:                  14.520 sec
   Reporting:                    0.700 sec
*******************************************************************************
```

Figure 4: Output of vpcluster for matrix *sports.mat* and a 10-way clustering that shows the descriptive and discriminating features of each cluster.

```
prompt% vpcluster -refine -rclassfile=sports.rclass -showfeatures sports.mat 10
*******************************************************************************
 CLUTO 1.5 - vpcluster, Copyright 2001-02, Regents of the University of Minnesota

Matrix Information ------------------------------------------------------------
  Name: sports.mat, #Rows: 8580, #Columns: 126373, #NonZeros: 1107980

Options -----------------------------------------------------------------------
  CRfun=I2, RowModel=None, ColModel=IDF, Prune=1.00, NTrials=10, NIter=10
  Direct=NO, RB=YES, Refine=YES, #Clusters: 10

Solution ----------------------------------------------------------------------

-------------------------------------------------------------------------------
10-way clustering solution: [I2=2.30e+03], Entropy: 0.169, Purity: 0.878
-------------------------------------------------------------------------------
cid  Size  ISim ISdev  ESim ESdev Entpy Purty | base bask foot hock boxi bicy golf
-------------------------------------------------------------------------------
  0   788 0.103 0.036 0.018 0.006 0.010 0.997 |    1    1    0  786    0    0    0
  1   682 0.109 0.035 0.021 0.006 0.011 0.997 |    1    0  680    0    0    0    1
  2   610 0.110 0.041 0.022 0.007 0.018 0.995 |  607    1    2    0    0    0    0
  3   818 0.090 0.034 0.020 0.007 0.014 0.996 |    0  815    2    1    0    0    0
  4   795 0.102 0.034 0.023 0.007 0.016 0.995 |  791    0    4    0    0    0    0
  5   863 0.103 0.038 0.025 0.009 0.000 1.000 |  863    0    0    0    0    0    0
  6   791 0.045 0.018 0.015 0.006 0.821 0.411 |   76   33   92    5  120  140  325
  7  1133 0.050 0.016 0.021 0.007 0.030 0.991 |    7    2 1123    1    0    0    0
  8   988 0.044 0.014 0.019 0.005 0.486 0.533 |   54  527  390    8    0    1    8
  9  1112 0.036 0.013 0.021 0.008 0.210 0.910 | 1012   31   53    8    2    4    2
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
10-way clustering solution - Descriptive & Discriminating Features...
-------------------------------------------------------------------------------
Cluster   0, Size:   788, ISim: 0.103, ESim: 0.018
      Descriptive:  col04688 22.5%, col00134  9.5%, col04423  4.3%, col02099  3.5%, col04483  1.6%
   Discriminating:  col04688 17.2%, col00134  6.0%, col04423  3.3%, col02099  2.3%, col04483  1.2%

Cluster   1, Size:   682, ISim: 0.109, ESim: 0.021
      Descriptive:  col00086 37.6%, col00091  8.1%, col00084  6.9%, col01091  2.8%, col00132  2.1%
   Discriminating:  col00086 28.6%, col00091  5.5%, col00084  5.2%, col01091  2.2%, col00132  1.5%

Cluster   2, Size:   610, ISim: 0.110, ESim: 0.022
      Descriptive:  col18174  9.1%, col11733  7.6%, col18183  6.4%, col01570  3.9%, col26743  3.3%
   Discriminating:  col18174  7.4%, col11733  5.9%, col18183  5.4%, col26743  2.7%, col01570  2.7%

Cluster   3, Size:   818, ISim: 0.090, ESim: 0.020
      Descriptive:  col02843 15.4%, col10737  4.3%, col06054  2.6%, col03655  2.4%, col03412  2.2%
   Discriminating:  col02843 12.7%, col10737  3.6%, col06054  2.2%, col03655  2.0%, col03412  1.7%

Cluster   4, Size:   795, ISim: 0.102, ESim: 0.023
      Descriptive:  col01536 20.6%, col04716  5.0%, col04640  3.4%, col03838  2.4%, col01045  2.1%
   Discriminating:  col01536 15.1%, col04716  4.3%, col04640  2.4%, col03838  2.1%, col01045  1.6%

Cluster   5, Size:   863, ISim: 0.103, ESim: 0.025
      Descriptive:  col04265  9.5%, col00281  8.3%, col13856  4.0%, col00340  4.0%, col01362  2.1%
   Discriminating:  col04265  6.3%, col00281  5.0%, col13856  2.5%, col00340  1.7%, col01362  1.4%

Cluster   6, Size:   791, ISim: 0.045, ESim: 0.015
      Descriptive:  col00351 31.6%, col01953  3.9%, col00396  3.0%, col00532  2.4%, col16968  2.1%
   Discriminating:  col00351 23.5%, col01953  3.4%, col00396  2.5%, col16968  1.8%, col00532  1.4%

Cluster   7, Size:  1133, ISim: 0.050, ESim: 0.021
      Descriptive:  col02393  3.5%, col10761  3.2%, col00031  2.6%, col00026  2.6%, col00024  2.4%
   Discriminating:  col02393  3.8%, col10761  3.4%, col00031  2.6%, col00026  2.6%, col00147  2.2%

Cluster   8, Size:   988, ISim: 0.044, ESim: 0.019
      Descriptive:  col01186  2.4%, col00910  2.3%, col00899  2.3%, col00616  1.9%, col00428  1.7%
   Discriminating:  col01186  2.2%, col00899  2.0%, col00910  1.8%, col00616  1.6%, col00611  1.4%

Cluster   9, Size:  1112, ISim: 0.036, ESim: 0.021
      Descriptive:  col01391  3.3%, col00169  2.3%, col01364  2.0%, col00606  1.9%, col01380  1.6%
   Discriminating:  col01391  3.9%, col00169  2.3%, col00606  1.8%, col01364  1.6%, col10638  1.2%
-------------------------------------------------------------------------------

Timing Information ------------------------------------------------------------
   I/O:                                1.540 sec
   Clustering:                        14.500 sec
   Reporting:                          0.700 sec
*******************************************************************************
```

Figure 5: Output of vpcluster for matrix *sports.mat* and a 10-way clustering that shows the descriptive and discriminating features of each cluster.

the resulting clustering solution at that point optimizes the specified clustering criterion function.

The format of the produced tree for the *sports.mat* data set is shown in Figure 6. This result was obtained by specifying both *-showtree* as well as the *-rclassfile* parameter that provides the class labels for each object in the matrix.

```
prompt% vpcluster -refine -rclassfile=sports.rclass -showtree sports.mat 10
*******************************************************************************
 CLUTO 1.5 - vpcluster, Copyright 2001-02, Regents of the University of Minnesota

Matrix Information ------------------------------------------------------------
  Name: sports.mat, #Rows: 8580, #Columns: 126373, #NonZeros: 1107980

Options -----------------------------------------------------------------------
  CRfun=I2, RowModel=None, ColModel=IDF, Prune=1.00, NTrials=10, NIter=10
  Direct=NO, RB=YES, Refine=YES, #Clusters: 10

Solution ----------------------------------------------------------------------

-------------------------------------------------------------------------------
10-way clustering solution: [I2=2.30e+03], Entropy: 0.169, Purity: 0.878
-------------------------------------------------------------------------------
cid  Size  ISim ISdev  ESim ESdev Entpy Purty | base bask foot hock boxi bicy golf
-------------------------------------------------------------------------------
  0   788 0.103 0.036 0.018 0.006 0.010 0.997 |    1    1    0  786    0    0    0
  1   682 0.109 0.035 0.021 0.006 0.011 0.997 |    1    0  680    0    0    0    1
  2   610 0.110 0.041 0.022 0.007 0.018 0.995 |  607    1    2    0    0    0    0
  3   818 0.090 0.034 0.020 0.007 0.014 0.996 |    0  815    2    1    0    0    0
  4   795 0.102 0.034 0.023 0.007 0.016 0.995 |  791    0    4    0    0    0    0
  5   863 0.103 0.038 0.025 0.009 0.000 1.000 |  863    0    0    0    0    0    0
  6   791 0.045 0.018 0.015 0.006 0.821 0.411 |   76   33   92    5  120  140  325
  7  1133 0.050 0.016 0.021 0.007 0.030 0.991 |    7    2 1123    1    0    0    0
  8   988 0.044 0.014 0.019 0.005 0.486 0.533 |   54  527  390    8    0    1    8
  9  1112 0.036 0.013 0.021 0.008 0.210 0.910 | 1012   31   53    8    2    4    2
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
Hierarchical Tree that optimizes the I2 criterion function...
-------------------------------------------------------------------------------
                          base bask foot hock boxi bicy golf
-------------------------------------------------------------
18
|-------14
|       |-------10
|       |       |---5      863    0    0    0    0    0    0
|       |       |---9     1012   31   53    8    2    4    2
|       |-13
|         |---------2      607    1    2    0    0    0    0
|         |---------4      791    0    4    0    0    0    0
|-17
  |---15
  |   |-------11
  |   |       |-----6       76   33   92    5  120  140  325
  |   |       |-----8       54  527  390    8    0    1    8
  |   |-----12
  |         |-------1        1    0  680    0    0    0    1
  |         |-------7        7    2 1123    1    0    0    0
  |-16
    |---------------0        1    1    0  786    0    0    0
    |---------------3        0  815    2    1    0    0    0
-------------------------------------------------------------
-------------------------------------------------------------------------------

Timing Information
-------------------------------------------------------------
   I/O:                                1.790 sec
   Clustering:                        14.760 sec
   Reporting:                          0.440 sec
*******************************************************************************
```

Figure 6: Output of vpcluster for matrix *sports.mat* that also shows the hierarchical tree built on top of the discovered clusters.

Looking at this figure we can see that vpcluster displays the tree in a rotated fashion, *i.e.*, the root of the tree is at the first column, and the tree grows from left to right. The leaves of this tree are numbered from 0 to *NClusters*-1, and each one represents the corresponding cluster discovered by vpcluster. The internal nodes are numbered from *NClusters* to 2**NClusters*-2, with the root being the highest numbered node. The numbering of the internal nodes is done so that nodes that were obtained by merging a pair of clusters at an earlier stage of the agglomerative process have lower numbers compared to nodes obtained at later stages. For example, in Figure 6 the node numbered 10 represents the first pair of clusters (5 and 9) that were merged, the node numbered 11 represents the second pair of clusters (6 and 8) that were merged, and so on.

In addition to the tree itself, vpcluster also prints information about how the objects of the various classes are distributed in each cluster. This information is identical to that presented in the earlier table, and are replicated here to provide a better understanding on the content of the clusters that are merged together. Thus, looking at the tree we can

see that the subtree rooted at node 14, contains clusters that primarily contain documents about baseball, whereas the subtree rooted at 12 primarily contain clusters whose documents are about football. If the *-rclassfile* was not specified, this information is omitted.

```
prompt% vpcluster -refine -rclassfile=sports.rclass -clabelfile=sports.clabel -showtree -labeltree sports.mat 10
*******************************************************************************
 CLUTO 1.5 - vpcluster, Copyright 2001-02, Regents of the University of Minnesota

Matrix Information ------------------------------------------------------------
  Name: sports.mat, #Rows: 8580, #Columns: 126373, #NonZeros: 1107980

Options -----------------------------------------------------------------------
  CRfun=I2, RowModel=None, ColModel=IDF, Prune=1.00, NTrials=10, NIter=10
  Direct=NO, RB=YES, Refine=YES, #Clusters: 10

Solution ----------------------------------------------------------------------

-------------------------------------------------------------------------------
10-way clustering solution: [I2=2.30e+03], Entropy: 0.169, Purity: 0.878
-------------------------------------------------------------------------------
cid  Size  ISim ISdev  ESim ESdev Entpy Purty | base bask foot hock boxi bicy golf
-------------------------------------------------------------------------------
  0   788 0.103 0.036 0.018 0.006 0.010 0.997 |    1    1    0  786    0    0    0
  1   682 0.109 0.035 0.021 0.006 0.011 0.997 |    1    0  680    0    0    0    1
  2   610 0.110 0.041 0.022 0.007 0.018 0.995 |  607    1    2    0    0    0    0
  3   818 0.090 0.034 0.020 0.007 0.014 0.996 |    0  815    2    1    0    0    0
  4   795 0.102 0.034 0.023 0.007 0.016 0.995 |  791    0    4    0    0    0    0
  5   863 0.103 0.038 0.025 0.009 0.000 1.000 |  863    0    0    0    0    0    0
  6   791 0.045 0.018 0.015 0.006 0.821 0.411 |   76   33   92    5  120  140  325
  7  1133 0.050 0.016 0.021 0.007 0.030 0.991 |    7    2 1123    1    0    0    0
  8   988 0.044 0.014 0.019 0.005 0.486 0.533 |   54  527  390    8    0    1    8
  9  1112 0.036 0.013 0.021 0.008 0.210 0.910 | 1012   31   53    8    2    4    2
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
Hierarchical Tree that optimizes the I2 criterion function...
-------------------------------------------------------------------------------
                          Size   ISim   XSim
18                     [ 8580, 0.026, 0.000] [giant  1.7%, yard  1.6%, hit  1.3%, box  1.2%, in  1.2%]
|-------14             [ 3380, 0.045, 0.017] [in  4.2%, giant  3.7%, hit  3.4%, pitch  2.3%, homer  2.0%]
|       |-------10     [ 1975, 0.049, 0.034] [in  4.9%, hit  4.7%, homer  2.2%, run  2.1%, sox  2.0%]
|       |       |---5  [  863, 0.103, 0.035] [in  9.5%, hit  8.3%, homer  4.0%, run  4.0%, sox  2.1%]
|       |       |---9  [ 1112, 0.036, 0.035] [basebal  3.3%, million  2.3%, pitcher  2.0%, brave  1.9%, leagu  1.6%]
|       |-13           [ 1405, 0.071, 0.034] [giant  9.7%, canseco  2.7%, pitch  2.4%, henderson  2.3%, mitchell  2.3%]
|         |---------2  [  610, 0.110, 0.036] [canseco  9.1%, henderson  7.6%, russa  6.4%, la  3.9%, mcgwire  3.3%]
|         |---------4  [  795, 0.102, 0.036] [giant 20.6%, mitchell  5.0%, craig  3.4%, mcgee  2.4%, clark  2.1%]
|-17                   [ 5200, 0.029, 0.017] [yard  3.9%, box  2.5%, shark  1.9%, goal  1.6%, warrior  1.4%]
  |---15               [ 3594, 0.031, 0.019] [yard  7.5%, box  4.1%, pass  2.1%, touchdown  1.4%, bowl  1.1%]
  |   |-------11       [ 1779, 0.031, 0.021] [box 10.0%, tournam  1.9%, santa  1.4%, confer  1.2%, golf  1.2%]
  |   |       |-----6  [  791, 0.045, 0.017] [box 31.6%, golf  3.9%, hole  3.0%, round  2.4%, par  2.1%]
  |   |       |-----8  [  988, 0.044, 0.017] [santa  2.4%, confer  2.3%, cal  2.3%, school  1.9%, st  1.7%]
  |   |-----12         [ 1815, 0.052, 0.021] [yard 15.1%, pass  4.0%, touchdown  3.0%, quarterback  1.7%, bowl  1.5%]
  |         |-------1  [  682, 0.109, 0.036] [yard 37.6%, pass  8.1%, touchdown  6.9%, td  2.8%, kick  2.1%]
  |         |-------7  [ 1133, 0.050, 0.036] [seifert  3.5%, montana  3.2%, raider  2.6%, nfl  2.6%, bowl  2.4%]
  |-16                 [ 1606, 0.060, 0.019] [shark  9.3%, warrior  6.0%, goal  4.5%, period  2.0%, score  1.9%]
    |---------------0  [  788, 0.103, 0.024] [shark 22.5%, goal  9.5%, nhl  4.3%, period  3.5%, penguin  1.6%]
    |---------------3  [  818, 0.090, 0.024] [warrior 15.4%, laker  4.3%, hardawai  2.6%, mullin  2.4%, nba  2.2%]
-------------------------------------------------------------------------------

Timing Information ------------------------------------------------------------
   I/O:                              1.830 sec
   Clustering:                      14.310 sec
   Reporting:                        0.900 sec
*******************************************************************************
```

Figure 7: Output of vpcluster for matrix *sports.mat* that shows the hierarchical tree built on top of the discovered clusters as well as the descriptive features of each cluster.

Besides showing the agglomerative tree, vpcluster can also analyze each of the clusters produced during this agglomerative process, displaying statistics regarding their quality and a set of descriptive features. This is done by specifying the *-labeltree* option. The output of vpcluster in this case is shown in Figure 7.

Looking at this figure we can see that in addition to the tree itself, vpcluster prints a number of statistics for each cluster. In particular, it displays the cluster's "Size" which is the number of objects in that cluster, the cluster's "ISim" which is the average similarity between the objects of each cluster, and the cluster's "XSim" which is the average similarity between the objects of each pair of clusters that are the children of the same node of the tree. For example, the cluster corresponding to node 13, contains 1405 documents, whose average similarity is 0.071, and the average similarity between the documents in this cluster and the documents in the cluster corresponding to node 10 is 0.034.

Next to these statistics, it prints the set of features that best describe each cluster. The method used to derive these features and the information that is displayed are identical to those used by the *-showfeatures* option.

3.3.5 Looking at the Visualizations

As discussed in Sections 3.1 and 3.2 both vpcluster and vacluster can produce a number of graphical visualizations showing the relation between the different objects, features, and clusters. Our goal in this section is to provide some illustrative examples of what the various *-plotXXX* commands can do.

Figure 8 shows the type of visualizations that can be produced when *-plotmatrix* is specified for a sparse matrix. In particular, Figure 8(a) shows the visualization produced by executing the following command:

```
vpcluster -plotmatrix=fig1.ps tr23.mat 10.
```

As we can see from that plot, vpcluster shows the rows of the input matrix re-ordered in such a way so that the rows assigned to each one of the ten clusters are numbered consecutively. The columns of the displayed matrix are selected to be the union of the *nfeatures* most descriptive and discriminating features of each cluster, and are ordered according their column-id. Also, at the top of each column, the label of each feature is shown (if you enlarge the postscript or PDF file of the manual you will be able to see the names of the words that these columns correspond to). Each non-zero positive element of the matrix is displayed by a different shade of red. Entries that are bright red correspond to large values and the brightness of the entries decreases as their value decrease. The values that are plotted correspond to the values obtained after applying the particular *-rowmodel* and *-colmodel*, and normalizing each row to be of unit length. Figure 8(b) shows a visualization of the same clustering solution in which the rows and the columns are also re-ordered according to a hierarchical clustering solution. In particular, this plot was obtained by executing the following command:

```
vpcluster -fulltree -clustercolumns -plotmatrix=fig2.ps tr23.mat 10.
```

As we can see from this plot, vpcluster now re-orders the rows and the columns so that rows/columns that are part of the same subtree are closer to each other in the final output. Also, along the rows and the columns of the displayed matrix, vpcluster draws the actual hierarchical tree that was computed. Finally, Figure 8(c) shows a visualization of the 10-way clustering solution obtained by vacluster. In particular, this plot was obtained by executing the following command:

```
vacluster -clustercolumns -plotmatrix=fig3.ps tr23.mat 10.
```

Figure 9 shows the type of visualizations that can be produced when *-plotmatrix* is specified for a dense matrix, for a particular micro-array gene expression data set. The three different visualizations were produced by executing the following commands, respectively:

```
vpcluster -denseinput -sim=2 -plotmatrix=fig4.ps genes1.mat 5
vpcluster -denseinput -sim=2 -fulltree -clustercolumns -plotmatrix=fig5.ps genes1.mat 5
vacluster -denseinput -sim=2 -clustercolumns -plotmatrix=fig6.ps genes1.mat 5
```

These plots are similar in nature to those produced for sparse matrices and the only difference is that they show all the columns (and not just the union of the descriptive and discriminating features). Also note that each row now has a label (corresponding to the name of the particular gene) that is read by default from the file name *"genes.mat.rlabel"*. Finally, note that the plots contain both red and green boxes, representing positive and negative values, respectively. The values used to derive the colors correspond to those used internally by CLUTO. In this particular example, since the clusters were obtained using the correlation coefficient, the values correspond to the mean-subtracted original row vectors, normalized to be of unit length.

A similar dense-matrix visualization is shown in Figure 10 for another micro-array gene expression data set. The different visualizations were produced by executing the following commands:

```
vacluster -denseinput -plotmatrix=fig7.ps genes2.mat 1
vacluster -denseinput -zeroblack -plotmatrix=fig8.ps genes2.mat 1
```

Figure 11 shows the type of visualization that can be produced when *-plotcluster* is specified for a sparse matrix. This plot was obtained by executing the following command:

```
vpcluster -clustercolumns -plotclusters=fig9.ps tr23.mat 10. vpcluster -clustercolumns -
   plotclusters=fig10.ps -nfeatures=10 -clabelfile=sports.cname sports.mat 20.
```

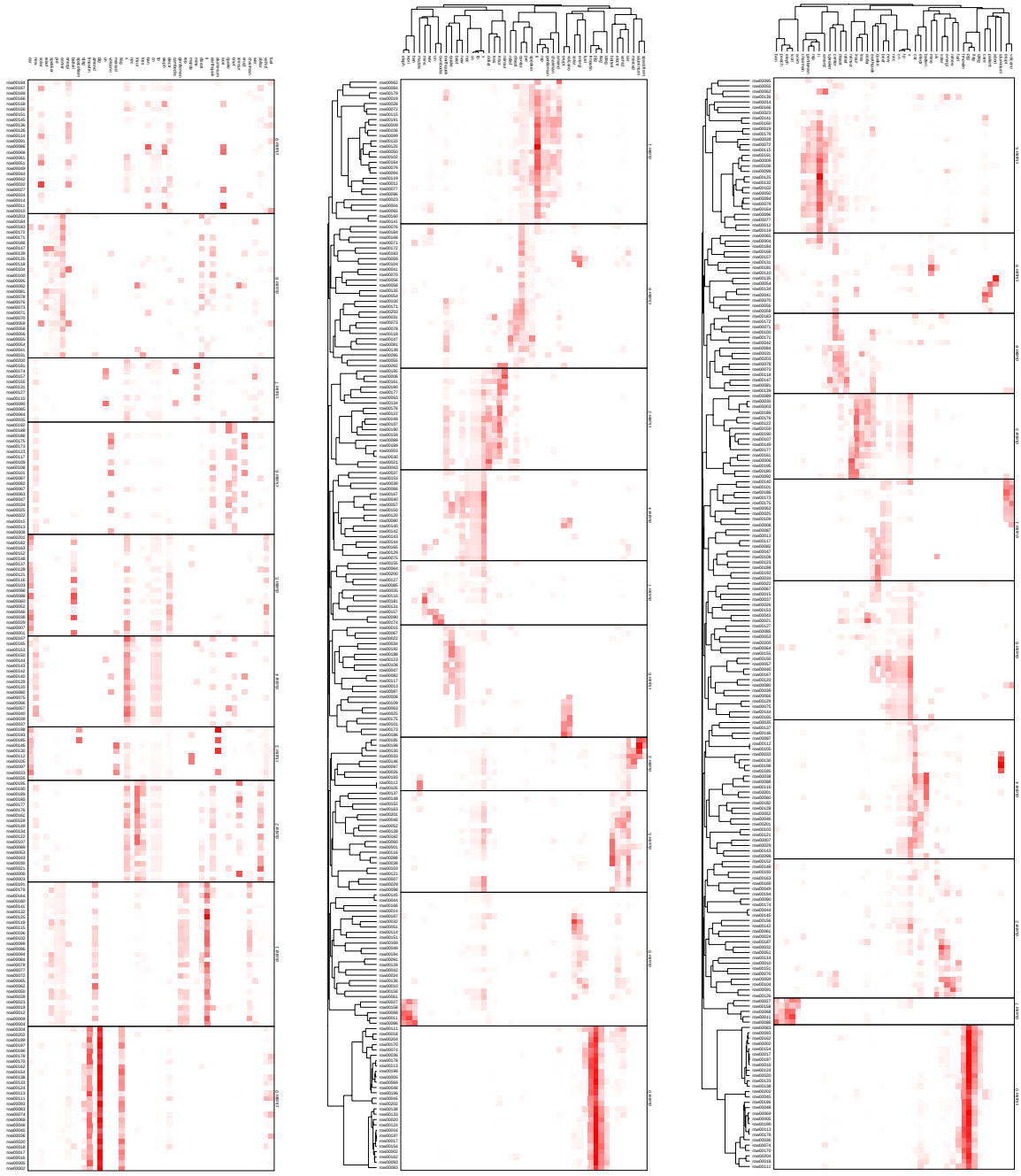


Figure 8: Various visualizations generated by the *-plotmatrix* parameter. (a) Shows the clustering solution produced by vpcluster; (b) Shows the same clustering solution but the rows and columns have been re-ordered. (c) Shows the clustering solution produced by vacluster.

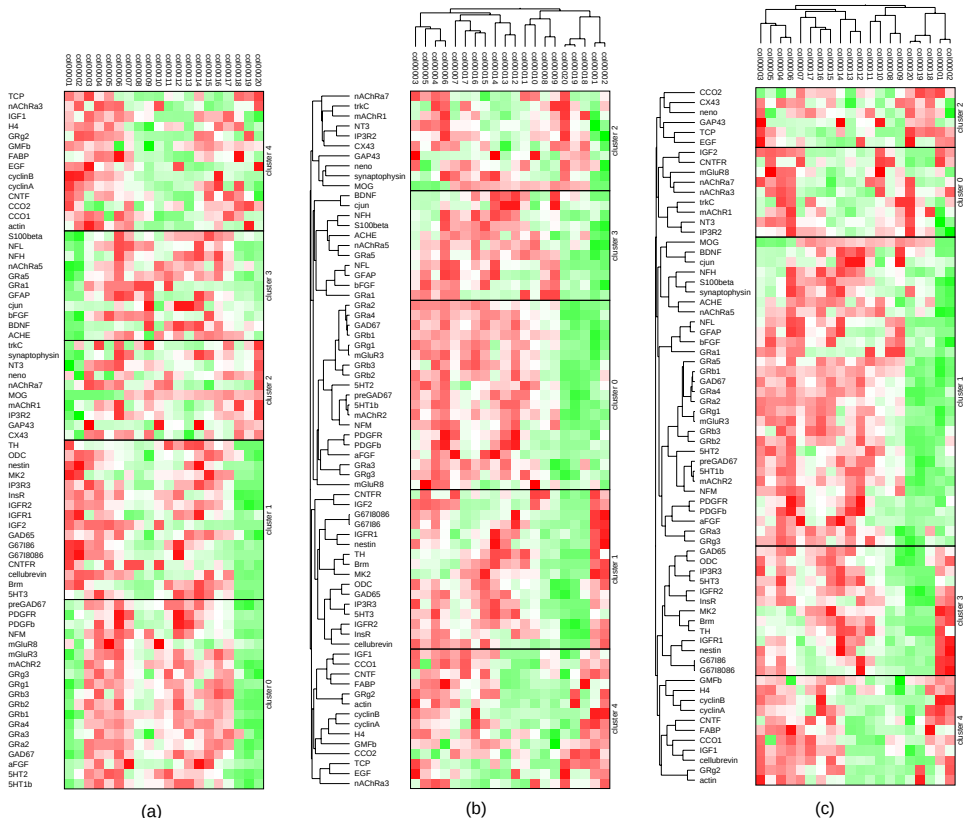


Figure 9: Various visualizations generated by the *-plotmatrix* parameter. (a) Shows the clustering solution produced by vpcluster; (b) Shows the same clustering solution but the rows and columns have been re-ordered. (c) Shows the clustering solution produced by vacluster.

This plot shows the clustering solution shown at Figure 8(b) by replacing the set of rows in each cluster by a single row that corresponds to the centroid vector of the cluster. The *-plotcluster* option is particularly useful for displaying very large data sets, as the number of rows in the plot is only equal to the number of clusters.

Finally, Figure 12 shows the type of visualization that can be produced when *-plottree* is specified. This plot was obtained by executing the following command:

```
vacluster -plottree=fig11.ps tr23.mat 10.
```

This plot shows the entire hierarchical tree for the *tr23.mat* data set. The leaves of the tree are labeled with the particular row-id (or row label if available). You can see the labels by properly magnifying the figure.

3.4 Input File Formats

The vpcluster program provided with CLUTO requires an input file that stores the objects to be clustered in a matrix format, as well as various optional files containing the column labels and the class labels of the various objects. The format of these files are described in the following sections.

3.4.1 Matrix File

The primary input of the programs in CLUTO is a matrix storing the objects to be clustered. Each row of this matrix represent a single object, and its various columns correspond to the dimensions (*i.e.*, features) of the objects. This matrix is stored in a file and is supplied to the various programs as one of the command line parameters.

CLUTO understands two different input matrix formats. The first format is suitable for sparse matrices and is the default format that is assumed by CLUTO; and the second format is suitable for storing dense matrices and is the

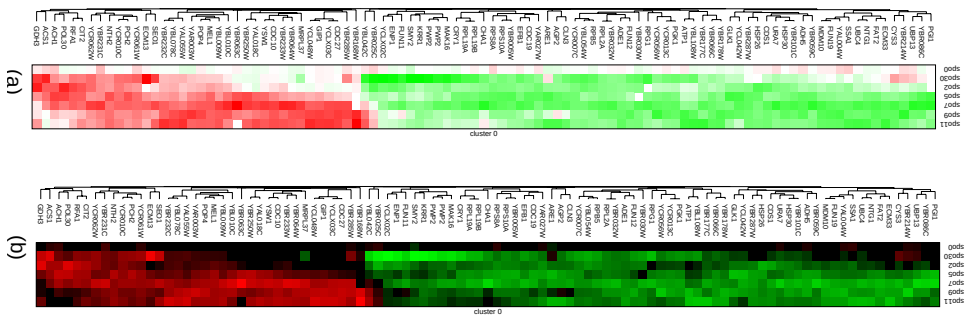


Figure 10: Various visualizations generated by the *-plotmatrix* parameter. (a) Shows the clustering solution produced by vacluster; (b) Shows the same clustering solution but the color scheme has been changed.

format that is assumed when the *-denseinput* parameter is specified.

Sparse Matrix Format A sparse matrix A with n rows and m columns is stored in a plain text file that contains $n+1$ lines. The first line contains information about the size of the matrix, while the remaining n lines contain information for each row of A. In CLUTO's sparse matrix format only the non-zero entries of the matrix are stored.

The first line of the matrix file contains exactly three numbers, all of which are integers. The first integer is the number of rows in the matrix (n), the second integer is the number of columns in the matrix (m), and the third integer is the total number of non-zeros entries in the $n \times m$ matrix.

The remaining n lines store information about the actual non-zero structure of the matrix. In particular, the $(i+1)$st line of the file contains information about the non-zero entries of the ith row of the matrix. Since the ith row corresponds to the ith object to be clustered, this is nothing more than the non-zero entries of the ith object's feature vector. The non-zero entries of each row are specified as a space-separated list of pairs. Each pair contains the column number followed by the value for that particular column (*i.e.*, feature). The column numbers are assumed to be integers and their corresponding values are assumed to be floating point numbers. The meaning of the values associated with each entry of the object's vector is problem dependent.

Note that the columns are numbered starting from 1 (not from 0 as is often done in C). Furthermore, CLUTO's matrix format does not require the column-pairs (column-number — column-value) to be sorted in any order.

An example of CLUTO's matrix format is shown in Figure 13. This figure shows an example 7×8 matrix and its corresponding representation in CLUTO's matrix format.

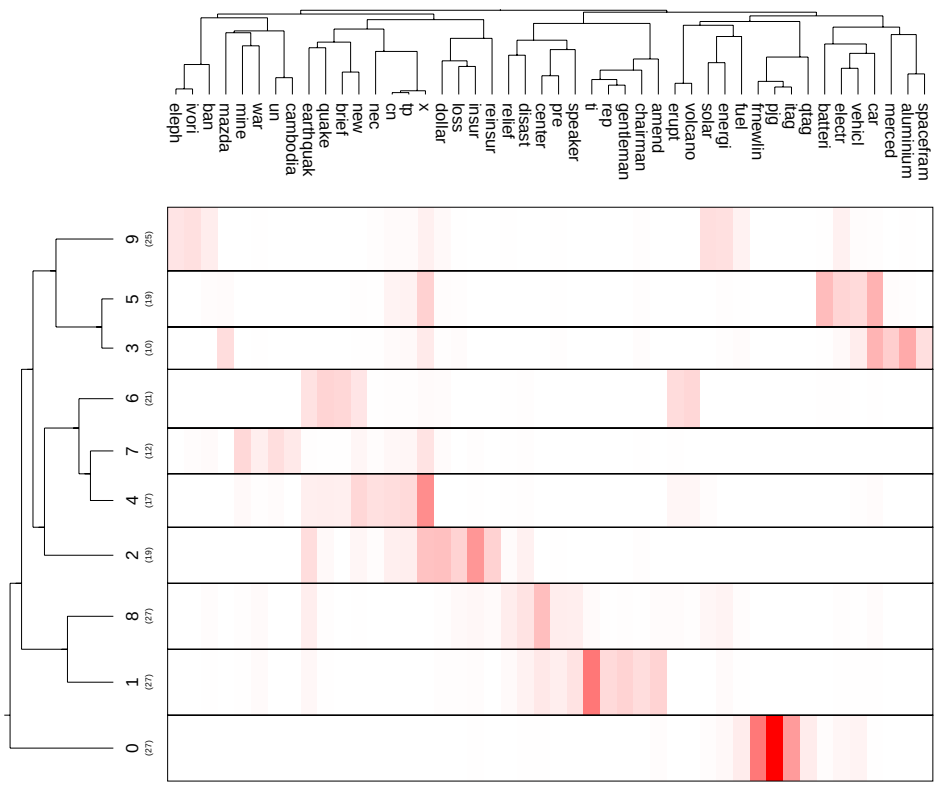
Dense Matrix Format A dense matrix A with n rows and m columns is stored in a plain text file that contains $n+1$ lines. The first line stores information about the size of the matrix, while the remaining n lines contain information for each row of A. The first line of the matrix file contains exactly two numbers, all of which are integers. The first integer is the number of rows in the matrix (n) and the second integer is the number of columns in the matrix (m). The remaining n lines store the values of the m columns for each one of the rows. In particular, each line contains exactly m space-separated floating point values, such that the ith value corresponds to the ith column of A.

3.4.2 Row Label File

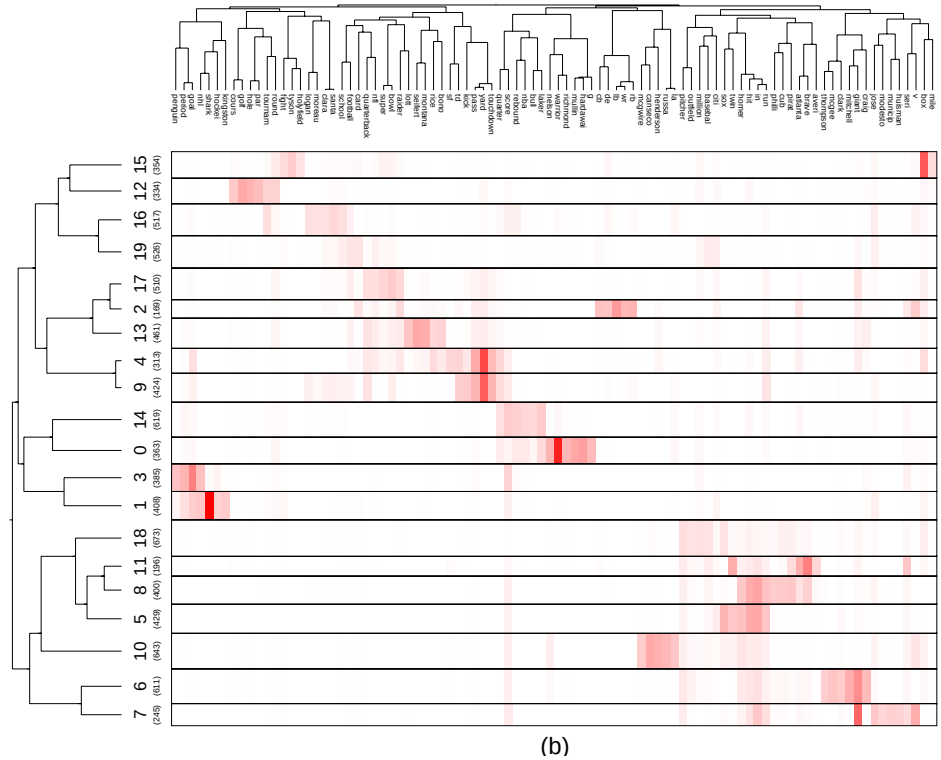
As discussed in Section 3, when the *-rlabelfile* parameter is used, CLUTO's stand-alone programs read a file that stores the label for each one of the rows (*i.e.*, objects) of the matrix. The format of this file is as follows. If n is the total number of rows in the matrix, then the row-label file contains exactly n lines. The information stored in each line is treated as a string and becomes the label of the corresponding row of the matrix. That is, the ith line of this file contains the label of the ith row of the matrix.

3.4.3 Column Label File

As discussed in Section 3.1, when the *-clabelfile* parameter is used, the vpcluster program reads a file that stores the label for each one of the columns (*i.e.*, features) of the matrix. The format of this file is as follows. If m is the total



(a)



(b)

Figure 11: Various visualizations generated by the *-plotcluster* parameter.

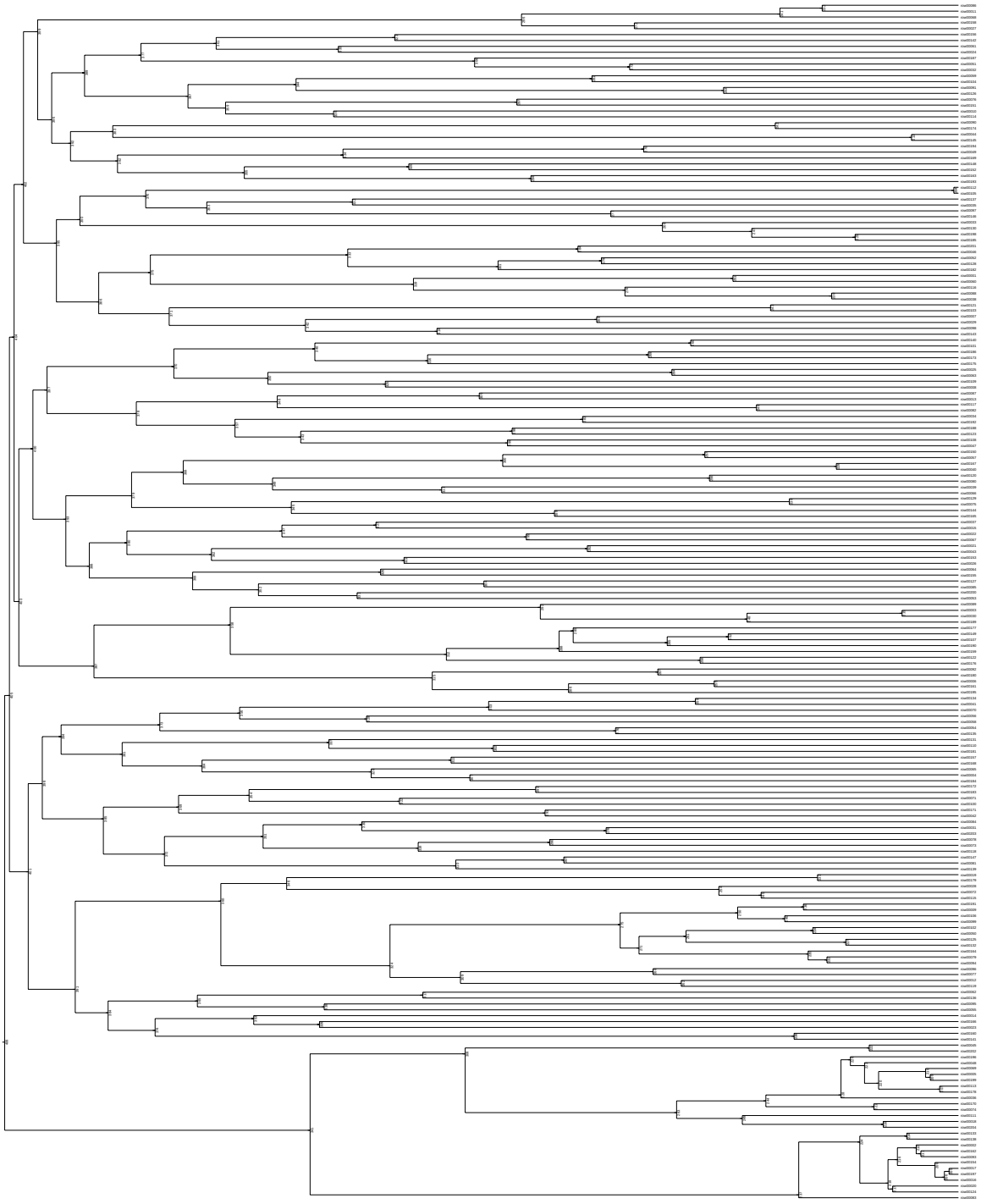


Figure 12: Various visualizations generated by the *-plottree* parameter.

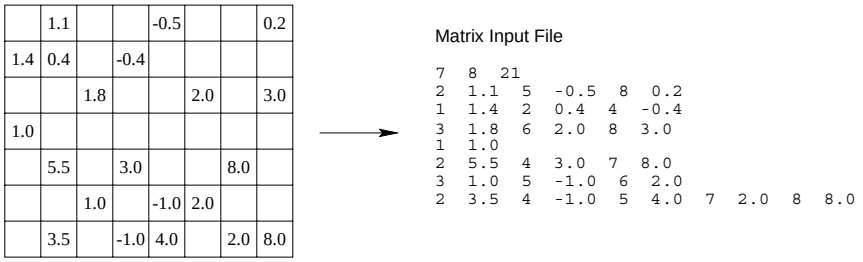


Figure 13: Storage format of a sample matrix.

number of columns in the matrix, then the column-label file contains exactly m lines. The information stored in each line is treated as a string and becomes the label of the corresponding column of the matrix. That is, the ith line of this file contains the label of the ith column of the matrix.

3.4.4 Row Class Label File

As discussed in Section 3.1, when the *-rclassfile* parameter is used, the vpcluster program reads a file that stores the class labels for each one of the rows (*i.e.*, objects) of the matrix. The format of this file is as follows. If n is the total number of rows in the matrix, then the class-label file contains exactly n lines. The information stored in each line is treated as a string and becomes the class-label of the corresponding object of the matrix. That is, the ith line of this file contains the label of the ith row of the matrix. In order to ensure that a set of objects belong to the same class, their corresponding rows in the class-label file must contain identical strings.

3.5 Output File Formats

The vpcluster program can generate two different output files. The first file contains the clustering vector and the internal and external z-scores for each object (when the *-zscores* option was specified), whereas the second file contains the hierarchical agglomerative tree that may be built on top of the computed clustering solution. The format of these files is described in the following sections.

3.5.1 Clustering Solution File

The clustering file of a matrix with n rows consists of n lines with a single number per line. The ith line of the file contains the cluster number that the ith object/row belongs to. Cluster numbers run from zero to the number of clusters minus one.

CLUTO's clustering algorithms remove all the columns that occur in fewer than three rows before computing the clustering solution. This is for performance reasons, and it does not affect the quality of the computed clustering solution. However, as a result of this pruning step, some objects may loose all of their features. In this case, CLUTO will not be able to assign these objects to any of the clusters. In this case, the cluster number for that particular row will be set to -1.

If the *-zscores* is specified, each line of this file also contains two additional numbers right after the cluster number. The first number is its internal z-score, and the second number is its external z-score.

3.5.2 Tree File

The tree produced by performing a hierarchical agglomerative clustering on top of the k-way clustering solution produced by vpcluster is stored in a file in the form of a *parent* array. In particular, if k is the number of clusters, then the *tree* file contains $2k-1$ lines, such that the ith line contains the parent of the ith node of the tree. In the case of the root node, that is stored in the last line of the file, the parent is set to -1. For example, the tree file for the tree shown in Figure 7 will contain 19 lines, and each line will store the following numbers (one number per line): 16, 12, 13, 16, 13, 10, 11, 12, 11, 10, 14, 15, 15, 14, 18, 17, 17, 18, -1.

In addition to the parent of each node, CLUTO's tree file also contains the average similarity between the siblings of

each tree node. Since this quantity is not defined for the leaves, only the rows of the file corresponding to the interior nodes of the tree contain meaningful numbers.

If for some reason, CLUTO's clustering programs cannot produce an entire single hierarchical tree, then the parent array will contain multiple subtrees. The subtrees can be re-constructed by traversing the parent array from the leaves toward the root. When a "-1" is encountered as the parent of a node other than the root's, then this particular subtree ends.

4 CLUTO's Library Interface

The functionality provided by CLUTO's vpcluster program can also be accessed directly from a C or C++ program by using the provided stand-alone library. In the rest of this section we provide information about how to link your program with CLUTO's library, describe the data structures used to pass information into the routines and give a detailed description of the calling sequence of the various routines.

4.1 Using CLUTO's Library

In order to use CLUTO's stand-alone library you must link your program with CLUTO's pre-compiled library that is provide in the software distribution. For Unix-based distributions, the name of the library is `libcluto.a`, and for the Windows 32 distribution, the name of the library file is `libcluto.lib`. At this point no dynamic link libraries are provided for either Unix- or Windows-based distributions; however, such libraries may be provided in the future.

The method by which an external library is linked to your program varies from system to system. In most Unix-based systems you can link it by just specifying `-lcluto` at the end of "cc" or "ld" command line. Care must be taken that CLUTO's library is in the default library search path. In most cases this can be modified by using the "-L" option to specify the directory where `libcluto.a` is stored. For Windows-based systems, the linking method depends on the particular development environment, and you should consult its documentation.

Any program that uses CLUTO's library must include the `cluto.h` header file that is provided with CLUTO's distribution. This file contains various constant definitions as well as function prototypes and allows C and C++ programs to access CLUTO's functions.

4.2 Matrix Data Structure

Most of the routines in CLUTO's library take, as input, the objects to be clustered in the form of a matrix. CLUTO's routines support both sparse and dense matrices using the same set of data structures. As with the matrix-file format of CLUTO's stand-alone programs, the rows of this matrix correspond to the objects, and the columns correspond to their various features.

Sparse Matrix Data Structure A sparse matrix is supplied to CLUTO's routines using a row-based compressed storage format (CSR). The CSR format is a widely used scheme for storing sparse matrices. In this format a matrix with n rows, m columns, and nnz non-zero entries is represented using three arrays that are called `rowptr`, `rowind`, and `rowval`. The array `rowptr` is of size $n+1$ whereas the arrays `rowind` and `rowval` are of size nnz.

The array `rowind` stores the column-indices of the non-zero entries in the matrix, and the array `rowval` stores their corresponding values. In particular, the array `rowind` stores the column-indices of the first row, followed by the column-indices of the second row, and so on. Similarly, the array `rowval` stores the corresponding values of the non-zero entries of the first row, followed by the corresponding values of the non-zero entries of the second row, and so on. The array `rowptr` is used to determine where the storage of a row starts and ends in the arrays, `rowind` and `rowval`. In particular, the column-indices of the ith row are stored starting at `rowind[rowptr[i]]` and ending at (but not including) `rowind[rowptr[i+1]]`. Similarly, the values of the non-zero entries of the ith row are stored starting at `rowval[rowptr[i]]` and ending at (but not including) `rowval[rowptr[i+1]]`. Also note that the number of non-zero entries of the ith row is simply `rowptr[i+1]-rowptr[i]`.

Figure 14 illustrates the CSR format for the sparse matrix used earlier to illustrated the format of the matrix file used by vpcluster. Note, that the numbering of the columns in the CSR format starts from zero and not from one.

Dense Matrix Data Structure A dense matrix is supplied to CLUTO's routines by using only the `rowval` array and setting the `rowptr` and `rowind` arrays to NULL. In fact, CLUTO's routines determine the input matrix format by checking to see if `rowptr` is NULL or not. A dense matrix with n rows and m columns is passed to CLUTO by supplying in `rowval` the $n \times m$ values of the matrix, in row-major order format. That is, the m values of the ith row (where i takes values from $0 \ldots n-1$) is stored starting at location `rowval[i*m]` and ending at (but not including) `rowval[(i+1)*m]`.

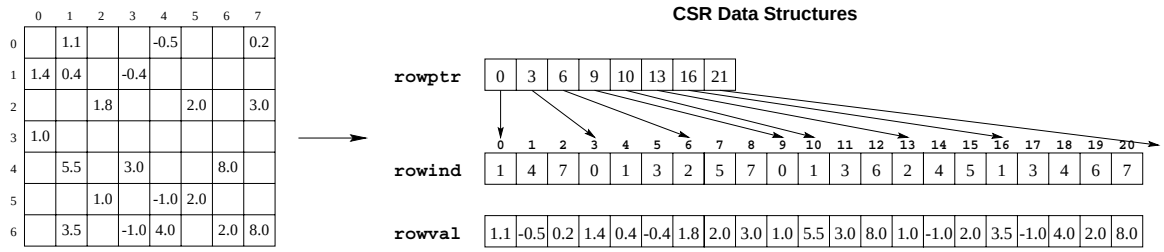


Figure 14: An example of the CSR format for storing sparse matrices.

4.3 Clustering Parameters

Most of CLUTO's routines take, as input, two parameters that control the similarity function to be used while clustering the objects and the clustering criterion function to be optimized in the process of clustering. These two parameters are called *simfun* and *crfun*, respectively.

4.3.1 The *simfun* Parameter

This parameter specified the similarity function to be used for clustering the objects. This parameter is similar to the *-sim* option of vpcluster and vacluster. The possible values for the *simfun* parameter are the following:

CLUTO_SIM_COSINE The similarity between the objects is computed using the cosine function of their vectors. This is the similarity function used by the default settings of vpcluster and vacluster.

CLUTO_SIM_CORRCOEF The similarity between the objects is computed using the correlation coefficient of their vectors.

4.3.2 The *crfun* Parameter

This parameter specifies the clustering criterion function to be used in finding the clusters. This parameter is similar to the *-crfun* option of vpcluster and vacluster. The possible values for the *crfun* parameter are the following:

CLUTO_CLFUN_I1 Selects the I1 criterion function ($\mathcal{I}_1$ in [2]).

CLUTO_CLFUN_I2 Selects the I2 criterion function ($\mathcal{I}_2$ in [2]). This is the criterion function used by the default settings of vpcluster and vacluster.

CLUTO_CLFUN_E1 Selects the E1 criterion function ($\mathcal{E}_1$ in [2]).

CLUTO_CLFUN_G1 Selects the G1 criterion function ($\mathcal{G}_1$ in [2]).

CLUTO_CLFUN_G1P Selects the G1' criterion function ($\mathcal{G}'_1$ in [2]).

CLUTO_CLFUN_H1 Selects the H1 criterion function ($\mathcal{H}_1$ in [2]).

CLUTO_CLFUN_H2 Selects the H2 criterion function ($\mathcal{H}_2$ in [2]).

4.4 Object Modeling Parameters

Most of CLUTO's routines take as input three parameters that control how the rows and columns of the input matrix will be modeled. These parameters are called *rowmodel*, *colmodel*, and *colprune*.

4.4.1 The *rowmodel* Parameter

This parameter specifies the model to be used for scaling the various columns of each row. This parameter is similar to the *-rowmodel* option of vpcluster and vacluster. The possible values for this parameter are:

CLUTO_ROWMODEL_NONE The columns of each row are not scaled and used as supplied in the `rowval` array. This is the default setting of vpcluster.

CLUTO_ROWMODEL_MAXTF The columns of each row are scaled so their values are between 0.5 and 1.0. This scaling scheme corresponds to vpcluster's scheme in which *-rowmodel=2*.

CLUTO_ROWMODEL_SQRT The columns of each row are scaled to be equal to the square root of their actual values. This scaling scheme corresponds to vpcluster's scheme in which *-rowmodel=3*.

CLUTO_ROWMODEL_LOG The columns of each row are scaled to be equal to the log of their actual values. This scaling scheme corresponds to vpcluster's scheme in which *-rowmodel=4*.

4.4.2 The *colmodel* Parameter

This parameter specifies the model to be used for scaling the various columns globally across all the rows of the matrix. This parameter is similar to the *-colmodel* option of vpcluster and vacluster. The possible values for this parameter are:

CLUTO_COLMODEL_NONE The columns of the matrix are not globally scaled and they are used as is.

CLUTO_COLMODEL_IDF The columns of the matrix are scaled according to the inverse document frequency paradigm (IDF), that was described in vpcluster's section. This is the default setting of vpcluster.

4.4.3 The *colprune* Parameter

This parameter specifies the factor by which the columns of the matrix will be pruned before performing the clustering. Valid range of values are from (0.0, 1.0]. A value of 1.0 indicates no pruning and is the default setting for vpcluster and vacluster.

4.5 Debugging Parameter

Most of CLUTO's routines take as input a parameter called *dbglvl* that controls the amount of information to be printed. The value is obtained by adding the codes for the various options. The possible options are:

CLUTO_DBG_PROGRESS Print information about the clustering process.

CLUTO_DBG_RPROGRESS Print information about the criterion optimization process as well about the progress of object-to-object similarity calculations.

CLUTO_DBG_APROGRESS Print information about the agglomeration process.

A value of zero inhibits any debugging output.

4.6 Clustering Routines

void **CLUTO_VP_ClusterDirect** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval, int simfun,
int crfun, int rowmodel, int colmodel, float colprune, int ntrials, int niter,
int seed, int dbglvl, int nclusters, int *part)

Description

Used to cluster a matrix into a specified (k) number of clusters using a partitional clustering algorithm that computes the k-way clustering directly. Provides the functionality of the *-direct* option of the vpcluster program.

Input Parameters

nrows, ncols

The number of rows and columns of the input matrix whose rows store the objects to be clustered.

rowptr, rowind, rowval

The matrix itself in the format described in Section 4.2.

simfun, crfun

The clustering parameters whose meaning and possible values are described in Section 4.3.

rowmodel, colmodel, colprune

The object modeling parameters whose meaning and possible values are described in Section 4.4.

ntrials Specifies the number of different clustering solutions to be computed. The solution that achieves the best value of the criterion function is the one that is returned. The value for *ntrials* must be greater than zero, and vpcluster's default setting is 10.

niter Specifies the maximum number of iterations that are performed during each refinement cycle. The value for *niter* has to be greater than zero.

seed The seed to be used by the random number generator.

dbglvl The debugging parameter whose meaning and possible values are described in Section 4.5.

nclusters The number of desired clusters.

Output Parameters

part This is an array of size *nrows* that upon successful completion stores the clustering vector of the matrix. The ith entry of this array stores the cluster number that the ith row of the matrix belongs to. Note that the numbering of the clusters starts from zero. The application is responsible for allocating the memory for this array.

Under certain circumstances, CLUTO may not be able to assign a particular row to a cluster. In this case, the *part[]* entry of that particular row will be set to -1.

Note

The various values for the *simfun*, *crfun*, *rowmodel*, *colmodel*, and *dbglvl* parameters are defined in `cluto.h`, and this header file must be included in all programs that use CLUTO's library.

void **CLUTO_VP_ClusterRB** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval,
int simfun, int crfun, int rowmodel, int colmodel, float colprune,
int ntrials, int niter, int seed, int kwayrefine, int dbglvl,
int nclusters, int *part)

Description

Used to cluster a matrix into a specified (k) number of clusters using a partitional clustering algorithm that computes the k-way by performing a sequence of repeated bisections. Provides the functionality of the *-rb* option of the vpcluster program.

Input Parameters

nrows, ncols

The number of rows and columns of the input matrix whose rows store the objects to be clustered.

rowptr, rowind, rowval

The matrix itself in the format described in Section 4.2.

simfun, crfun

The clustering parameters whose meaning and possible values are described in Section 4.3.

rowmodel, colmodel, colprune

The object modeling parameters whose meaning and possible values are described in Section 4.4.

ntrials Specifies the number of different clustering solutions to be computed. The solution that achieves the best value of the criterion function is the one that is returned. The value for *ntrials* must be greater than zero.

niter Specifies the maximum number of iterations that are performed during each refinement cycle. The value for *niter* has to be greater than zero.

seed The seed to be used by the random number generator.

kwayrefine

This parameter controls whether or not the clustering solution will be globally optimized at the end by performing a series of k-way refinement iterations. The possible values for this parameter are:

0 Does not optimize the clustering solution globally.

1 Optimizes the clustering solution globally.

The global optimization of the clustering solution can significantly increase the amount of time required to perform the clustering.

dbglvl The debugging parameter whose meaning and possible values are described in Section 4.5.

nclusters The number of desired clusters.

Output Parameters

part This is an array of size *nrows* that upon successful completion stores the clustering vector of the matrix. The ith entry of this array stores the cluster number that the ith row of the matrix belongs to. Note that the numbering of the clusters starts from zero. The application is responsible for allocating the memory for this array.

Under certain circumstances, CLUTO may not be able to assign a particular row to a cluster. In this case, the *part[]* entry of that particular row will be set to -1.

Note

The various values for the *simfun*, *crfun*, *rowmodel*, *colmodel*, and *dbglvl* parameters are defined in `cluto.h`, and this header file must be included in all programs that use CLUTO's library.

CLUTO_VP_ClusterRB is considerably faster than CLUTO_VP_ClusterDirect and it should be preferred if the number of desired clusters is quite large (*e.g.*, greater than 20–30).

void **CLUTO_VA_Cluster** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval,
int simfun, int crfun, int rowmodel, int colmodel, float colprune,
int dbglvl, int nclusters, int *part, int *ptree, float *tsims)

Description

Used to cluster a matrix into a specified (k) number of clusters using a hierarchical agglomerative clustering algorithm. Provides the functionality of the vacluster program.

Input Parameters

nrows, ncols
The number of rows and columns of the input matrix whose rows store the objects to be clustered.

rowptr, rowind, rowval
The matrix itself in the format described in Section 4.2.

simfun, crfun
The clustering parameters whose meaning and possible values are described in Section 4.3.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 4.4.

dbglvl The debugging parameter whose meaning and possible values are described in Section 4.5.

nclusters The number of desired clusters.

Output Parameters

part This is an array of size *nrows* that upon successful completion stores the clustering vector of the matrix. The *i*th entry of this array stores the cluster number that the *i*th row of the matrix belongs to. Note that the numbering of the clusters starts from zero. The application is responsible for allocating the memory for this array.

Under certain circumstances, CLUTO may not be able to assign a particular row to a cluster. In this case, the *part[]* entry of that particular row will be set to -1.

ptree This is an array of size *2*nrows* that upon successful completion stores the parent array of the binary hierarchical tree. In this tree, each node corresponds to a cluster. The leaf nodes are the original *nrows* objects, and they are numbered from 0 to *nrows*-1. The internal nodes of the tree are numbered from *nrows* to 2**nrows*-2. The numbering of the internal nodes is performed so that smaller numbers correspond to clusters obtained by merging a pair of clusters earlier during the agglomeration process. The root of the tree is numbered 2**nrows*-2.

The *i*th entry of the *ptree* array stores the parent node of the *i* node of the tree. The *ptree* entry for the root is set to -1.

The application is responsible for allocating the memory for this array.

tsims This is an array of size *2*nrows* that upon successful completion stores the average similarity between every pair of siblings in the induced tree. In particular, *tsims[i]* stores the average pairwise similarity between the pair of clusters that are the children of the *i*th node of the tree. Note that the first *nrows* entries of this vector are not defined and are set to 0.0.

The application is responsible for allocating the memory for this array.

Note

The various values for the *simfun*, *crfun*, *rowmodel*, *colmodel*, and *dbglvl* parameters are defined in `cluto.h`, and this header file must be included in all programs that use CLUTO's library.

Due to the high computational requirements of CLUTO_VA_Cluster, it should only be used to cluster matrices that have fewer than 3,000–6,000 rows.

void **CLUTO_SA_Cluster** (int nobjects, int *objwgts, float *smat, int simfun, int crfun, int memflag,
int dbglvl, int nclusters, int *part, int *ptree, float *tsims, float *gains)

Description

Used to cluster a set of objects into a specified (k) number of clusters using a hierarchical agglomerative clustering algorithm. Unlike the rest of CLUTO's clustering routines, CLUTO_SA_Cluster uses a user-supplied object-to-object similarity matrix.

Input Parameters

nobjects The number of objects to be clustered.

objwgts This is an array of size *nobjects* that stores the weight of each object. For normal operation supply an array whose entries are set to one.

smat An array of size *nobjects*$\times$*nobjects* that stores the similarity between any pair of objects. The similarity between the *i*th and *j*th objects is stored at location `smat[i*nobjects+j]` and `smat[j*nobjects+i]`. The between object similarities have to be symmetric, and the similarity between the *i*th object with itself must also be provided.

crfun The clustering criterion function whose meaning and possible values are described in Section 4.3.2.

memflag This parameter controls whether or not CLUTO_SA_Cluster can overwrite the memory that *smat* points to. The possible values for this parameter are:

CLUTO_MEM_NOREUSE It does not overwrite the original *smat* array.

CLUTO_MEM_REUSE It overwrites the original *smat* array.

If CLUTO_MEM_NOREUSE is specified, CLUTO_SA_Cluster allocates sufficient memory to make a copy of *smat*, which increases the overall memory complexity of the clustering algorithm.

dbglvl The debugging parameter whose meaning and possible values are described in Section 4.5.

nclusters The number of desired clusters.

Output Parameters

part An array of size *nobjects* that upon successful completion stores the clustering vector of the dataset. The *i*th entry of this array stores the cluster number that the *i*th object belongs to. Note that the numbering of the clusters starts from zero. The application is responsible for allocating the memory for this array.

ptree An array of size *2*nobjects* that upon successful completion stores the parent array of the binary hierarchical tree. In this tree, each node corresponds to a cluster. The leaf nodes are the original objects, and they are numbered from 0 to *nobjects*-1. The internal nodes of the tree are numbered from *nobjects* to 2**nobjects*-2. The numbering of the internal nodes is done in such a fashion so that smaller numbers correspond to clusters obtained by merging a pair of clusters earlier during the agglomeration process. The root of the tree is numbered 2**nobjects*-2.

The *i*th entry of the *ptree* array stores the parent node of the *i* node of the tree. The *ptree* entry for the root is set to -1.

The application is responsible for allocating the memory for this array.

tsims An array of size *2*nobjects* that upon successful completion stores the average similarity between every pair of siblings in the induced tree. In particular, *tsims[i]* stores the average pairwise similarity between the pair of clusters that are the children of the *i*th node of the tree. Note that the first *nobjects* entries of this vector are not defined and are set to 0.0.

The application is responsible for allocating the memory for this array.

gains An array of size *2*nobjects* that upon successful completion stores the gains in the value of the criterion function obtained by merging every pair of siblings in the induced tree. In particular, *gains[i]* stores the gain achieved by merging the pair of clusters that are the children of the *i*th node of the tree. Note that the first *nobjects* entries of this vector are not defined.

The application is responsible for allocating the memory for this array.

Note

The various values for the *crfun*, *memflag*, and *dbglvl* parameters are defined in `cluto.h`, and this header file must be included in all programs that use CLUTO's library.

Due to the high computational requirements of CLUTO_SA_Cluster, it should only be used to cluster datasets with fewer than 3,000–6,000 objects.

void **CLUTO_V_BuildTree** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval,
int simfun, int crfun, int rowmodel, int colmodel, float colprune,
int treetype, int nclusters, int *part, int *ptree, float *tsims)

Description

Builds a hierarchical agglomerative tree that preserves the clustering solution supplied in the *part* array. It can build two types of trees. The first type is a tree built on top of a particular clustering solution, such that the leaves of the tree correspond to the different clusters. This is the type of tree used when the *-showtree* option of vpcluster and vacluster is specified. The second type of tree is a complete agglomerative tree that preserves the clustering. This is the type of tree used when the *-fulltree* option of vpcluster is specified. The hierarchical agglomerative tree is build so that it optimizes a particular clustering criterion function.

Input Parameters

nrows, ncols

The number of rows and columns of the input matrix whose rows store the objects to be clustered.

rowptr, rowind, rowval

The matrix itself in the format described in Section 4.2.

simfun, crfun

The clustering parameters whose meaning and possible values are described in Section 4.3.

rowmodel, colmodel, colprune

The object modeling parameters whose meaning and possible values are described in Section 4.4.

treetype

Specifies the type of tree that needs to be built. The possible values for this parameter are:

CLUTO_TREE_TOP	Builds a tree whose leaves correspond to the different clusters.
CLUTO_TREE_FULL	Builds a complete tree that preserves the clustering solution.

nclusters The number of clusters in the supplied clustering solution.

part An array of size *nrows* that stores the clustering solution. The *i*th entry of this array stores the cluster number that the *i*th row of the matrix belongs to. This array should correspond to a clustering solution returned by CLUTO's clustering routines. Note that the numbering of the clusters starts from zero.

Output Parameters

ptree An array whose size depends on the type of tree that is requested.

If *treetype==CLUTO_TREE_TOP*, then it is of size *2*nclusters* that upon successful completion stores the parent array of the binary hierarchical tree. In this tree, each node corresponds to a cluster. The leaf nodes are the original *nclusters* clusters supplied via the *part* array, and they are numbered from 0 to *nclusters*-1. The internal nodes of the tree are numbered from *nclusters* to 2**nclusters*-2. The root of the tree is numbered 2**nclusters*-2.

If *treetype==CLUTO_TREE_FULL*, then it is of size *2*nrows* that upon successful completion stores the parent array of the binary hierarchical tree. In this tree, each node corresponds to a cluster. The leaf nodes are the original rows of the matrix, and they are numbered from 0 to *nrows*-1. The internal nodes of the tree are numbered from *nrows* to 2**nrows*-2. The root of the tree is numbered 2**nrows*-2.

The numbering of the internal nodes is done in such a fashion so that smaller numbers correspond to clusters obtained by merging a pair of clusters earlier during the agglomeration process.

The *i*th entry of the *ptree* array stores the parent node of the *i* node of the tree. The *ptree* entry for the root is set to -1.

The application is responsible for allocating the memory for this array.

tsims An array whose size depends on the type of tree that is requested. If *treetype==CLUTO_TREE_TOP*, then it is of size *2*nclusters* and if *treetype==CLUTO_TREE_FULL* then it is of size *2*nrows*.

Upon successful completion stores the average similarity between every pair of siblings in the induced tree. In particular, *tsims[i]* stores the average pairwise similarity between the pair of clusters that are the children of the *i*th node of the tree. Note that the first *nclusters* or *nrows* entries of this vector are not defined and are set to 0.0.

The application is responsible for allocating the memory for this array.

Note

The various values for the *simfun*, *crfun*, *rowmodel*, *colmodel*, and *treetype* parameters are defined in `cluto.h`, and this header file must be included in all programs that use CLUTO's library.

In order for this routine to build the accurate tree for a particular clustering solution, the values for the *rowmodel*, *colmodel*, and *colprune* parameters should be identical to those used to compute the clustering solution.

This routine can be used to build the hierarchical agglomerative tree with respect to any clustering criterion function regardless of the criterion function used to compute the clustering solution.

4.7 Cluster Statistics Routines

float **CLUTO_V_GetSolutionQuality** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval, int simfun, int crfun, int rowmodel, int colmodel, float colprune, int nclusters, int *part)

Description

Returns the value of a particular criterion function for a given clustering solution.

Input Parameters

nrows, ncols
The number of rows and columns of the input matrix whose rows store the objects that were clustered.

rowptr, rowind, rowval
The matrix itself in the format described in Section 4.2.

simfun, crfun
The clustering parameters whose meaning and possible values are described in Section 4.3.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 4.4.

nclusters The number of clusters in the supplied clustering solution.

part An array of size *nrows* that stores the clustering solution. The *i*th entry of this array stores the cluster number that the *i*th row of the matrix belongs to. This array should correspond to a clustering solution returned by CLUTO's clustering routines. Note that the numbering of the clusters starts from zero.

Returned Value

This function returns the value of the clustering criterion function of the supplied clustering solution. Please refer to [2] for the exact definitions of these criterion functions.

Note

The various values for the *simfun*, *crfun*, *rowmodel*, and *colmodel* parameters are defined in `cluto.h`, and this header file must be included in all programs that use CLUTO's library.

This routine can be used to find the value of any clustering criterion function regardless of the criterion function used to compute the clustering solution.

void **CLUTO_V_GetClusterStats** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval,
int simfun, int rowmodel, int colmodel, float colprune, int nclusters,
int *part, int *pwgts, float *cintsim, float *cintsdev, float *izscores,
float *cextsim, float *cextsdev, float *ezscores)

Description

Returns a number of statistics about a given clustering solution.

Input Parameters

nrows, ncols
The number of rows and columns of the input matrix whose rows store the objects that were clustered.

rowptr, rowind, rowval
The matrix itself in the format described in Section 4.2.

simfun The clustering similarity function whose meaning and possible values are described in Section 4.3.1.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 4.4.

nclusters The number of clusters in the supplied clustering solution.

part An array of size *nrows* that stores the clustering solution. The *i*th entry of this array stores the cluster number that the *i*th row of the matrix belongs to. This array should correspond to a clustering solution returned by CLUTO's clustering routines. Note that the numbering of the clusters starts from zero.

Output Parameters

pwgts An array of size *nclusters* that returns the sizes of the different clusters. In particular, the size of the *i*th cluster is returned in pwgts[*i*]. The application is responsible for allocating the memory for this array.

cintsim An array of size *nclusters* that returns the average similarity between the objects assigned to each cluster. In particular, the average similarity between the objects of the *i*th cluster is returned in cintsim[*i*]. The application is responsible for allocating the memory for this array.

cintsdev An array of size *nclusters* that returns the standard deviation of the average similarity between each object and the other objects in its own cluster. In particular, the standard deviation of the *i*th cluster is returned in cintsdev[*i*]. The application is responsible for allocating the memory for this array.

izscores An array of size *nrows* that returns the internal z-scores of each object. The internal z-score of the *i*th object is returned in izscores[*i*]. The internal z-score of each object is described in the discussion of the *-zscores* option of vpcluster. The application is responsible for allocating the memory for this array.

cextsim An array of size *nclusters* that returns the average similarity between the objects of each cluster and the remaining objects. In particular, the average *external* similarity of the objects of the *i*th cluster is returned in cextsim[*i*]. The application is responsible for allocating the memory for this array.

cextsdev An array of size *nclusters* that returns the standard deviation of the average external similarities of each object. In particular, the *external* standard deviation of the objects of the *i*th cluster is returned in cextsdev[*i*]. The application is responsible for allocating the memory for this array.

ezscores An array of size *nrows* that returns the external z-scores of each object. The external z-score of the *i*th object is returned in ezscores[*i*]. The external z-score of each object is described in the discussion of the *-zscores* option of vpcluster. The application is responsible for allocating the memory for this array.

Note

The various values for the *simfun*, *rowmodel*, and *colmodel* parameters are defined in `cluto.h`, and this header file must be included in all programs that use CLUTO's library.

In order for this routine to get the accurate statistics for a particular clustering solution, the values for the *rowmodel*, *colmodel*, and *colprune* parameters should be identical to those used to compute the clustering solution.

void **CLUTO_V_GetClusterFeatures** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval,
int simfun, int rowmodel, int colmodel, float colprune, int nclusters,
int *part, int nfeatures, int *internalids, float *internalwgts,
int *externalids, float *externalwgts)

Description

Returns the set of features that best describe and discriminate each one of the clusters of a given clustering solution. It provides the functionality of the *-showfeatures* option of the vpcluster program.

Input Parameters

nrows, ncols
The number of rows and columns of the input matrix whose rows store the objects that were clustered.

rowptr, rowind, rowval
The matrix itself in the format described in Section 4.2.

simfun The clustering similarity function whose meaning and possible values are described in Section 4.3.1.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 4.4.

nclusters The number of clusters in the supplied clustering solution.

part This is an array of size *nrows* that stores the clustering solution. The ith entry of this array stores the cluster number that the ith row of the matrix belongs to. This array should correspond to a clustering solution returned by CLUTO's clustering routines. Note that the numbering of the clusters starts from zero.

nfeatures The number of descriptive and discriminating features that is desired.

Output Parameters

internalids
An array of size *nclusters*nfeatures* that returns the column numbers of the ***descriptive*** features. The set of features of the ith cluster are stored in the *internalids* array starting at location $i * nfeatures$ up to location (but excluding) $(i + 1) * nfeatures$. The set of features for each cluster are returned in decreasing importance order. The numbering of the returned columns starts from zero. The application is responsible for allocating the memory for this array.

internalwgts
An array of size *nclusters*nfeatures* that returns the weight of each one of the descriptive features returned in the *internalids* array. The weight of the features stored in the ith location of the *internalids* array is returned in the ith location of the *internalwgts* array. The weights are numbers between 0.0 and 1.0 and represent the fraction of the within cluster similarity that each particular feature is responsible for. The application is responsible for allocating the memory for this array.

externalids
This is an array of size *nclusters*nfeatures* that returns the column numbers of the ***discriminating*** features. The set of features of the ith cluster are stored in the *externalids* array starting at location $i * nfeatures$ up to location (but excluding) $(i + 1) * nfeatures$. The set of features for each cluster are returned in decreasing importance order. The numbering of the returned columns starts from zero. The application is responsible for allocating the memory for this array.

externalwgts
This is an array of size *nclusters*nfeatures* that returns the weight of each one of the discriminating features returned in the *externalids* array. The weight of the features stored in the ith location of the *externalids* array is returned in the ith location of the *externalwgts* array. The weights are numbers between 0.0 and 1.0 and represent the fraction of the dissimilarity between the cluster and the rest of the objects that each particular feature is responsible for. The application is responsible for allocating the memory for this array.

Note

The various values for the *simfun*, *rowmodel*, and *colmodel* parameters are defined in `cluto.h`, and this header file must be included in all programs that use CLUTO's library.

In order for this routine to get the accurate set of features for a particular clustering solution, the values for the *rowmodel*, *colmodel*, and *colprune* parameters should be identical to those used to compute the clustering solution.

void **CLUTO_V_GetTreeStats** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval,
int simfun, int rowmodel, int colmodel, float colprune, int nclusters,
int *part, int *ptree, int *pwgts, float *cintsim, float *cextsim)

Description

Returns a number of statistics about the clusters corresponding to the different nodes of the hierarchical agglomerative tree.

Input Parameters

nrows, ncols
The number of rows and columns of the input matrix whose rows store the objects that were clustered.

rowptr, rowind, rowval
The matrix itself in the format described in Section 4.2.

simfun The clustering similarity function whose meaning and possible values are described in Section 4.3.1.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 4.4.

nclusters The number of clusters in the supplied clustering solution.

part An array of size *nrows* that stores the clustering solution. The *i*th entry of this array stores the cluster number that the *i*th row of the matrix belongs to. This array should correspond to a clustering solution returned by CLUTO's clustering routines. Note that the numbering of the clusters starts from zero.

ptree An array of size *2*nclusters* that was populated by the CLUTO_V_BuildTree routine.

Output Parameters

pwgts An array of size *2*nclusters* that returns the sizes of the clusters corresponding to the various nodes of the tree. In particular, the size of the cluster corresponding to the *i*th tree-node is returned in pwgts[*i*]. The application is responsible for allocating the memory for this array.

cintsim An array of size *2*nclusters* that returns the average similarity between the objects assigned to each cluster. In particular, the average similarity between the objects of the cluster corresponding to the *i*th tree-node is returned in cintsim[*i*]. The application is responsible for allocating the memory for this array.

cextsim An array of size *2*nclusters* that returns the average similarity between the objects of each cluster and their sibling cluster in the tree. In particular, the average *external* similarity of the objects of the *i*th cluster is returned in cextsim[*i*]. Note that each pair of sibling clusters will have the same *cextsim* value. The application is responsible for allocating the memory for this array.

Note

The various values for the *simfun*, *rowmodel*, and *colmodel* parameters are defined in `cluto.h`, and this header file must be included in all programs that use CLUTO's library.

In order for this routine to get the accurate statistics for a particular clustering solution, the values for the *rowmodel*, *colmodel*, and *colprune*, *nclusters*, *part*, and *ptree* parameters should be identical to those used to compute the clustering solution and build the hierarchical agglomerative tree.

void **CLUTO_V_GetTreeFeatures** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval,
int simfun, int rowmodel, int colmodel, float colprune, int nclusters,
int *part, int *ptree, int nfeatures, int *internalids, float *internalwgts,
int *externalids, float *externalwgts)

Description

Returns the set of features that best describe and discriminate each one of the clusters corresponding to the various nodes of the hierarchical agglomerative tree that was built on top of the clustering solution. It provides the functionality of the *-labeltree* option of the vpcluster program.

Input Parameters

nrows, ncols

The number of rows and columns of the input matrix whose rows store the objects that were clustered.

rowptr, rowind, rowval

The matrix itself in the format described in Section 4.2.

simfun The clustering similarity function whose meaning and possible values are described in Section 4.3.1.

rowmodel, colmodel, colprune

The object modeling parameters whose meaning and possible values are described in Section 4.4.

nclusters The number of clusters in the supplied clustering solution.

part An array of size *nrows* that stores the clustering solution. The ith entry of this array stores the cluster number that the ith row of the matrix belongs to. This array should correspond to a clustering solution returned by CLUTO's clustering routines. Note that the numbering of the clusters starts from zero.

ptree An array of size *2*nclusters* that was populated by the CLUTO_V_BuildTree routine.

nfeatures The number of descriptive and discriminating features that is desired.

Output Parameters

internalids

An array of size *2*nclusters*nfeatures* that returns the column numbers of the ***descriptive*** features. The set of features of the cluster corresponding to the ith tree node are stored in the *internalids* array starting at location $i * nfeatures$ up to location (but excluding) $(i+1) * nfeatures$. The set of features for each cluster are returned in decreasing importance order. The numbering of the returned columns starts from zero. The application is responsible for allocating the memory for this array.

internalwgts

An array of size *2*nclusters*nfeatures* that returns the weight of each one of the descriptive features returned in the *internalids* array. The weight of the features stored in the ith location of the *internalids* array is returned in the ith location of the *internalwgts* array. The weights are numbers between 0.0 and 1.0 and represent the fraction of the within cluster similarity that each particular feature is responsible for. The application is responsible for allocating the memory for this array.

externalids

An array of size *2*nclusters*nfeatures* that returns the column numbers of the ***discriminating*** features. The discriminating features are defined within the context of the pair of clusters that are the children of the same tree node. Consequently, there are no discriminating features for the root node of the tree. The set of features of the cluster corresponding to the ith tree node are stored in the *externalids* array starting at location $i * nfeatures$ up to location (but excluding) $(i+1) * nfeatures$. The set of features for each cluster are returned in decreasing importance order. The numbering of the returned columns starts from zero. The application is responsible for allocating the memory for this array.

externalwgts

An array of size *2*nclusters*nfeatures* that returns the weight of each one of the discriminating features returned in the *externalids* array. The weight of the features stored in the *i*th location of the *externalids* array is returned in the *i*th location of the *externalwgts* array. The weights are numbers between 0.0 and 1.0 and represent the fraction of the dissimilarity between the cluster and the rest of the objects that each particular feature is responsible for. The application is responsible for allocating the memory for this array.

Note

The various values for the *simfun*, *rowmodel*, and *colmodel* parameters are defined in `cluto.h`, and this header file must be included in all programs that use CLUTO's library.

In order for this routine to get the accurate set of features for a particular clustering solution, the values for the *rowmodel*, *colmodel*, and *colprune*, *nclusters*, *part*, and *ptree* parameters should be identical to those used to compute the clustering solution and build the hierarchical agglomerative tree.

5 System Requirements and Contact Information

CLUTO is written in ANSI C and has been extensively tested under Linux, Solaris, and Windows. At this point CLUTO's distribution is only in a binary format, as it is actively under development. However, we expect to make the source code available in future releases.

Even though, CLUTO contains no known bugs, it does not mean that all of its bugs have been found and fixed. If you find any problems, please send email to *karypis@cs.umn.edu*, with a brief description of the problem you have found. Also, any future updates to vpcluster will be made available on WWW at *http://www.cs.umn.edu/~karypis/cluto*.

6 Copyright Notice and Usage Terms

References

[1] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley & Sons, 2001.

[2] Ying Zhao and George Karypis. Criterion functions for document clustering: Experiments and analysis. Technical Report TR #01–40, Department of Computer Science, University of Minnesota, Minneapolis, MN, 2001. Available on the WWW at *http://cs.umn.edu/~karypis/publications*.