CLUTO *
A Clustering Toolkit

Release 2.1.1

George Karypis

karypis@cs.umn.edu

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

Technical Report: #02-017

November 28, 2003

 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

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 low and high dimensional datasets and for analyzing the characteristics of the various clusters.

CLUTO provides three different classes of clustering algorithms that operate either directly in the object's feature space or in the object's similarity space. These algorithms are based on the partitional, agglomerative, and graph-partitioning paradigms. A key feature in most of CLUTO's clustering algorithms is that they treat the clustering problem as an optimization process which seeks to maximize or minimize a particular ***clustering criterion function*** defined either globally or locally over the entire clustering solution space. CLUTO provides a total of seven different criterion functions that can be used to drive both partitional and agglomerative clustering algorithms, that are described and analyzed in [6, 5]. Most of these criterion functions have been shown to produce high quality clustering solutions in high dimensional datasets, especially those arising in document clustering. In addition to these criterion functions, CLUTO provides some of the more traditional local criteria (*e.g.*, single-link, complete-link, and UPGMA) that can be used in the context of agglomerative clustering. Furthermore, CLUTO provides graph-partitioning-based clustering algorithms that are well-suited for finding clusters that form contiguous regions that span different dimensions of the underlying feature space.

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 [6]. CLUTO's graph-partitioning-based clustering algorithms utilize high-quality and efficient multilevel graph partitioning algorithms derived from the METIS and hMETIS graph and hypergraph partitioning algorithms [4, 3].

CLUTO also 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. 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 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.

CLUTO's distribution consists of both stand-alone programs (vcluster and scluster) 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 Outline of CLUTO's Manual

CLUTO's manual is organized as follows. Section 3 describes the stand-alone programs provided by CLUTO, and discusses its various options and analysis capabilities. Section 4 describes the type of clusters that CLUTO's algorithms can find, and discusses their scalability. Section 5 describes the application programming interface (API) of the stand-alone library that implements the various algorithms implemented in CLUTO. Finally, Section 6 describes the system requirements for the CLUTO package.

2 Major Changes From Release 2.0

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

1. CLUTO provides a new class of ***biased agglomerative*** clustering algorithms that use a partitional clustering solution to bias the agglomeration process. The key motivation behind these algorithms is to use a partitional clustering solution that optimizes a global criterion function to limit the number of errors performed during the early stages of the agglomerative algorithms. Extensive experiments with these algorithms on document datasets show that they lead to superior clustering solutions [5].

2. CLUTO provides a new method for analyzing the discovered clusters and identify the set of features that co-occur within the objects of each cluster. This functionality is provided via the new *-showsummaries* parameter.

3. CLUTO provides a new method for selecting the cluster to be bisected next in the context of partitional clustering algorithms based on repeated bisectioning. This method that is specified by selecting *-cstype=largess* is based on analyzing the set of dimensions (*i.e.*, subspace) that account for the bulk of the similarity of each cluster, and selecting the cluster that leads to the largest decrease of these dimensions. This approach was motivated by the observation that in high-dimensional datasets, good clusters are embedded in low-dimensional subspaces.

4. CLUTO's graph partitioning algorithms can now compute the similarity between objects using the extended Jaccard coefficient that takes into account both the direction and the magnitude of the object vectors. Experiments with high-dimensional datasets arising in commercial and document domains showed that this similarity function is better than cosine-based similarity.

3 Using CLUTO via its Stand-Alone Program

CLUTO provides access to its various clustering and analysis algorithms via the vcluster and scluster stand-alone programs. The key difference between these programs is that vcluster takes as input the actual multi-dimensional representation of the objects that need to be clustered (*i.e.*, "v" comes from *vector*), whereas scluster takes as input the similarity matrix (or graph) between these objects (*i.e.*, "s" comes from *similarity*). Besides this difference, both programs provide similar functionality.

The rest of this section describes how to use these programs, how to interpret their output, the format of the various input files they require, and the format of the output files they produce.

3.1 The vcluster and scluster Clustering Programs

The vcluster and scluster programs are used to cluster a collection of objects into a predetermined number of clusters k. The vcluster program treats each object as a vector in a high-dimensional space, and it computes the clustering solution using one of five different approaches. Four of these approaches are *partitional* in nature, whereas the fifth approach is *agglomerative*. On the other hand, the scluster program operates on the similarity space between the objects and can compute the overall clustering solution using the same set of five different approaches.

Both the vcluster and scluster programs are invoked by providing two required parameters on the command line along with a number of optional parameters. Their overall calling sequence is as follows:

vcluster	[optional parameters]	*MatrixFile*	*NClusters*
scluster	[optional parameters]	*GraphFile*	*NClusters*

MatrixFile is the name of the file that stores the n objects to be clustered. In vcluster, 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 3.3.1. Similarly, *GraphFile*, is the name of the file that stores the adjacency matrix of the similarity graph between the n objects to be clustered. The exact format of the graph-file is described in Section 3.3.2. The second argument for both programs, *NClusters*, is the number of clusters that is desired.

Upon successful execution, vcluster and scluster display 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*** (or ***GraphFile.clustering.NClusters***), whose format is described in Section 3.4.1.

The behavior of vcluster and scluster can be controlled by specifying a number of different optional parameters (described in subsequent sections). These parameters can be broadly categorized into three groups. The first group controls various aspects of the clustering algorithm, the second group controls the type of analysis and reporting that is performed on the computed clusters, and the third set controls the visualization of the 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.

Examples of Using vcluster and scluster Figure 1 shows the output of vcluster for clustering a matrix into 10 clusters. From this figure we see that vcluster 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 the subsequent sections), and the number of desired clusters (*#Clusters*). Once it computes the clustering solution, it displays information regarding the quality of the overall clustering solution and the quality of each cluster. The meaning of the various measures that are reported will be discussed in Section 3.2. Finally, vcluster reports the time taken by the various phases of the program. For this particular example, vcluster required 0.950 seconds to read the input file and write the clustering solution, 9.060 seconds to compute the actual clustering solution, and 0.240 seconds to compute statistics on the quality of the clustering.

Similarly, Figure 2 shows the output of scluster for clustering a different dataset into 10 clusters. In this example

```
prompt% vcluster sports.mat 10
*******************************************************************************
vcluster (CLUTO 2.1) Copyright 2001-02, Regents of the University of Minnesota

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

Options ----------------------------------------------------------------------
  CLMethod=RB, CRfun=I2, SimFun=Cosine, #Clusters: 10
  RowModel=None, ColModel=IDF, GrModel=SY-DIR, NNbrs=40
  Colprune=1.00, EdgePrune=-1.00, VtxPrune=-1.00, MinComponent=5
  CSType=Best, AggloFrom=0, AggloCRFun=I2, NTrials=10, NIter=10

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

------------------------------------------------------------------------
10-way clustering: [I2=2.29e+03] [8580 of 8580]
------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  |
------------------------------------------------------------------------
  0   359 +0.168 +0.050 +0.020 +0.005 |
  1   629 +0.106 +0.041 +0.022 +0.007 |
  2   795 +0.102 +0.036 +0.018 +0.006 |
  3   762 +0.099 +0.034 +0.021 +0.006 |
  4   482 +0.098 +0.045 +0.022 +0.009 |
  5   844 +0.095 +0.035 +0.023 +0.007 |
  6  1724 +0.059 +0.026 +0.022 +0.007 |
  7  1175 +0.051 +0.015 +0.021 +0.006 |
  8   853 +0.043 +0.015 +0.019 +0.006 |
  9   957 +0.032 +0.012 +0.015 +0.006 |
------------------------------------------------------------------------

Timing Information -----------------------------------------------------------
   I/O:                                   0.950 sec
   Clustering:                            9.060 sec
   Reporting:                             0.240 sec
*******************************************************************************
```

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

the similarity between the objects was computed as the cosine between the object vectors. From this figure we see that scluster initially prints information about the graph, such as its name, the number of vertices (*#vtxs*), and the number of edges in the graph (*#Edges*). Next it prints information about the values of the various options that it used to compute the clustering, and the number of desired clusters (*#Clusters*). Once it computes the clustering solution, it displays information regarding the quality of the overall clustering solution and the quality of each cluster. Finally, scluster reports the time taken by the various phases of the program. For this particular example, scluster required 12.930 seconds to read the input file and write the clustering solution, 34.730 seconds to compute the actual clustering solution, and 0.610 seconds to compute statistics on the quality of the clustering. Note that even though the dataset used by scluster contained only 3204 objects, it took almost $3\times$ more time than that required by vcluster to cluster a dataset with 8580 objects. The performance difference between these two approaches is due to the fact that scluster operates on the graph that in this example contains almost 3204^2 edges.

3.1.1 Clustering Algorithm Parameters

There are a total of 18 different optional parameters that control how vcluster and scluster compute the clustering solution. The name and function of these parameters is described in the rest of this section. Note for each parameter we also list the program(s) for which they are applicable.

-clmethod=string **vcluster & scluster**

This parameter selects the method to be used for clustering the objects. The possible values are:

rb In this method, the desired k-way clustering solution is computed 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, the cluster is bisected so that the resulting 2-way clustering solution optimizes a particular clustering criterion function (which is selected using the *-crfun* parameter). Note that this approach ensures that the criterion function is locally optimized within each bisection, but in general is not globally optimized. The cluster that is selected for further partitioning is controlled by the *-cstype* parameter. By default, vcluster uses this approach to find the k-way clustering solution.

rbr In this method the desired k-way clustering solution is computed in a fashion similar to the

```
prompt% scluster la1.graph 10
*******************************************************************************
scluster (CLUTO 2.1) Copyright 2001-02, Regents of the University of Minnesota

Graph Information -------------------------------------------------------------
  Name: la1.graph, #Vtxs: 3204, #Edges: 10252448

Options -----------------------------------------------------------------------
  CLMethod=RB, CRfun=I2, #Clusters: 10
  EdgePrune=-1.00, VtxPrune=-1.00, GrModel=SY-DIR, NNbrs=40, MinComponent=5
  CSType=Best, AggloFrom=0, AggloCRFun=I2, NTrials=10, NIter=10

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

------------------------------------------------------------------------
10-way clustering: [I2=6.59e+02] [3204 of 3204]
------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  |
------------------------------------------------------------------------
  0    93 +0.128 +0.045 +0.013 +0.003 |
  1   261 +0.083 +0.025 +0.013 +0.003 |
  2   214 +0.048 +0.024 +0.015 +0.005 |
  3   191 +0.043 +0.014 +0.013 +0.004 |
  4   285 +0.040 +0.015 +0.013 +0.004 |
  5   454 +0.036 +0.015 +0.013 +0.005 |
  6   302 +0.035 +0.015 +0.011 +0.004 |
  7   307 +0.027 +0.009 +0.012 +0.004 |
  8   504 +0.027 +0.010 +0.014 +0.005 |
  9   593 +0.032 +0.013 +0.012 +0.004 |
------------------------------------------------------------------------

Timing Information ------------------------------------------------------------
   I/O:                                    12.930 sec
   Clustering:                             34.730 sec
   Reporting:                               0.610 sec
*******************************************************************************
```

Figure 2: Output of scluster for graph *la1.graph* and a 10-way clustering.

repeated-bisecting method but at the end, the overall solution is globally optimized. Essentially, vcluster uses the solution obtained by *-clmethod=rb* as the initial clustering solution and tries to further optimize the clustering criterion function.

direct In this method, the desired k-way clustering solution is computed 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 10–20), the direct approach leads to better clusters than those obtained via repeated bisections. However, as k increases, the repeated-bisecting approach tends to be better than direct clustering.

agglo In this method, the desired k-way clustering solution is computed using the ***agglomerative*** paradigm whose goal is to locally optimize (minimize or maximize) a particular clustering criterion function (which is selected using the *-crfun* parameter). The solution is obtained by stopping the agglomeration process when k clusters are left.

graph In this method, the desired k-way clustering solution is computed by first modeling the objects using a nearest-neighbor graph (each object becomes a vertex, and each object is connected to its most similar other objects), and then splitting the graph into k-clusters using a min-cut graph partitioning algorithm. **Note that if the graph contains more than one connected component, then vcluster and scluster return a $(k+m)$-way clustering solution, where m is the number of connected components in the graph.**

bagglo In this method, the desired k-way clustering solution is computed in a fashion similar to the *agglo* method; however, the agglomeration process is biased by a partitional clustering solution that is initially computed on the dataset. When *bagglo* is used, CLUTO first computes a $\sqrt{n}$-way clustering solution using the *rb* method, where n is the number of objects to be clustered. Then, it augments the original feature space by adding $\sqrt{n}$ new dimensions, one for each cluster. Each object is then assigned a value to the dimension corresponding to its own cluster, and this value is proportional to the similarity between that object and its cluster-centroid. Now, given this augmented representation, the overall clustering solution is obtained by using the traditional agglomerative paradigm and the clustering criterion function selected using the

-crfun parameter. The solution is obtained by stopping the agglomeration process when k clusters are left. Our experiments on document datasets, showed that this *biased* agglomerative approach always outperformed the traditional agglomerative algorithms [5].

The suitability of these clustering methods are in general domain and application dependent. Section 4 discusses relative merits of the various methods and their scalability characteristics. Also, you can refer to [6, 5] (which are included with CLUTO' distribution) for a detailed comparisons of the *rb*, *rbr*, *direct*, *agglo*, and *bagglo* approaches in the context of clustering document datasets.

-sim=string **vcluster**

Selects the similarity function to be used for clustering. The possible values are:

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

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

dist The similarity between objects is computed to be inversely proportional to the Euclidean distance between the objects. This similarity function is only applicable when *-clmethod=graph*.

jacc The similarity between objects is computed using the extended Jaccard coefficient. This similarity function is only applicable when *-clmethod=graph*.

The runtime of vcluster may increase for *-sim=corr*, as it needs to store and operate on the dense $n \times m$ matrix.

-crfun=string **vcluster & scluster**

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:

i1 Selects the I1 criterion function.

i2 Selects the I2 criterion function. This is the default setting for the *rb*, *rbr*, and *direct* clustering methods.

e1 Selects the E1 criterion function.

g1 Selects the G1 criterion function.

g1p Selects the G1′ criterion function.

h1 Selects the H1 criterion function.

h2 Selects the H2 criterion function.

slink Selects the traditional single-link criterion function.

wslink Selects a cluster-weighted single-link criterion function.

clink Selects the traditional complete-link criterion function.

wclink Selects a cluster-weighted complete-link criterion function.

upgma Selects the traditional UPGMA criterion function. This is the default setting for the *agglo* and *bagglo* clustering methods.

The precise mathematical definition of the first seven functions is shown in Table 1. The reader is referred to [6] for both a detailed description and evaluation of the various criterion functions. The *slink*, *wslink*, *clink*, *wclink*, and *upgma* criterion functions can only be used within the context of agglomerative clustering, and cannot be used for partitional clustering.

The *wslink* and *wclink* criterion function were designed for building an agglomerative solution on top of an existing clustering solution (see *-agglofrom*, or *-showtree* options). In this context, the weight of the "link" between two clusters S_i and S_j is set equal to the aggregate similarity between the objects of S_i to the objects in S_j divided by the total similarity between the objects in $S_i \bigcup S_j$.

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.

Note that the computational complexity of the agglomerative clustering algorithms (*i.e.*, *-clmethod=agglo* or *-clmethod=bagglo*) depend on the criterion function that is selected. In particular, if n is the number of objects, the complexity for $\mathcal{H}_1$ and $\mathcal{H}_2$ criterion functions is $O(n^3)$, whereas the complexity of the remaining criterion functions is $O(n^2 \log n)$. The higher 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.

Criterion Function	Optimazition Function
$\mathcal{I}_1$	$$\text{maximize} \quad \sum_{i=1}^{k} \frac{1}{n_i} \left(\sum_{v,u \in S_i} \text{sim}(v, u) \right) \tag{1}$$
$\mathcal{I}_2$	$$\text{maximize} \quad \sum_{i=1}^{k} \sqrt{\sum_{v,u \in S_i} \text{sim}(v, u)} \tag{2}$$
$\mathcal{E}_1$	$$\text{minimize} \quad \sum_{i=1}^{k} n_i \frac{\sum_{v \in S_i, u \in S} \text{sim}(v, u)}{\sqrt{\sum_{v,u \in S_i} \text{sim}(v, u)}} \tag{3}$$
$\mathcal{G}_1$	$$\text{minimize} \quad \sum_{i=1}^{k} \frac{\sum_{v \in S_i, u \in S} \text{sim}(v, u)}{\sum_{v,u \in S_i} \text{sim}(v, u)} \tag{4}$$
$\mathcal{G}'_1$	$$\text{minimize} \quad \sum_{i=1}^{k} n_i^2 \frac{\sum_{v \in S_i, u \in S} \text{sim}(v, u)}{\sum_{v,u \in S_i} \text{sim}(v, u)} \tag{5}$$
$\mathcal{H}_1$	$$\text{maximize} \quad \frac{\mathcal{I}_1}{\mathcal{E}_1} \tag{6}$$
$\mathcal{H}_2$	$$\text{maximize} \quad \frac{\mathcal{I}_2}{\mathcal{E}_1} \tag{7}$$

Table 1: The mathematical definition of CLUTO's clustering criterion functions. The notation in these equations are as follows: k is the total number of clusters, S is the total objects to be clustered, S_i is the set of objects assigned to the ith cluster, n_i is the number of objects in the ith cluster, v and u represent two objects, and sim(v, u) is the similarity between two objects.

-agglofrom=int **vcluster & scluster**

This parameter instructs the clustering programs to compute a clustering by combining both the partitional and agglomerative methods. In this approach, the desired k-way clustering solution is computed by first clustering the dataset into m clusters ($m > k$), and then the final k-way clustering solution is obtained by merging some of these clusters using an agglomerative algorithm. The number of clusters m is the input to this parameter. The method used to obtained the agglomerative solution is controlled by the *-agglocrfun* parameter.

This approach was motivated by the two-phase clustering approach of the CHAMELEON algorithm [2], and was designed to allow the user to compute a clustering solution that uses a different clustering criterion function for the partitioning phase from that used for the agglomeration phase. An application of such an approach is to allow the clustering algorithm to find non-globular clusters. In this case, the partitional clustering solution can be computed using a criterion function that favors globular clusters (*e.g.*, 'i2'), and

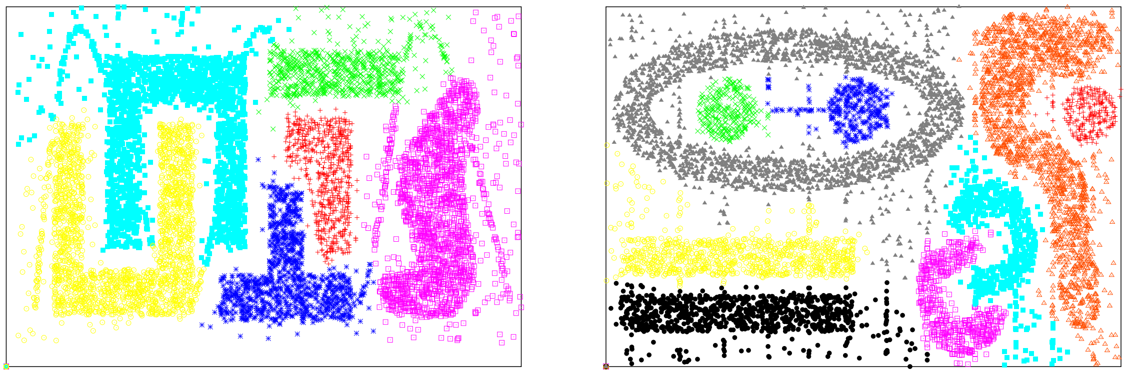


Figure 3: Examples of using the *-agglofrom* option for two spatial datasets. The result in (a) was obtained by running 'vcluster t4.mat 6 -clmethod=graph -sim=dist -agglofrom=30' and the results in (b) was obtained by running 'vcluster t7.mat 9 -clmethod=graph -sim=dist -agglofrom=30'.

then combine these clusters using a single-link approach (*e.g.*, 'wslink') to find non-globular but well-connected clusters. Figure 3 shows two such examples for two 2D point datasets.

-agglocrfun=string **vcluster & scluster**

This parameter controls the criterion function that is used during the agglomeration when the *-agglofrom* or the *-fulltree* option was specified. The values that this parameter can take are identical to those used by the *-crfun* parameter. If *-agglocrfun* is not specified, then for the partitional clustering methods it uses the same criterion function as that used to find the clusters, for the agglomerative methods it uses UPGMA, and for the graph-partitioning-based clustering methods, it uses the "wslink" criterion function.

-cstype=string **vcluster & scluster**

This parameter selects the method that is used to select the cluster to be bisected next when *-clmethod* is equal to "rb", "rbr", or "graph". The possible values are:

large Selects the largest cluster to be bisected next.

best Selects the cluster whose bisection will optimize the value of the overall clustering criterion function the most. This is the default option.

Note that in the case of graph-partitioning based clustering, the overall criterion function is evaluated in terms of the ratio cut, as to prevent (up to a point) the creation of very small clusters. However, this method is not 100% robust, so if you notice that in your dataset you are getting a clustering solution that contains very large and very small clusters, you should use "large" instead.

largess Selects the cluster that will lead to the larger reduction on the number of dimensions of the feature-space that account for the majority of the within-cluster similarity of the objects. This reduction in the *subspace-size* is weighted by the size of each cluster, as well. This method is applicable only to vcluster, and it should be used mostly with sparse and high dimensional datasets.

-fulltree **vcluster & scluster**

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 *-agglocrfun*). 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. Note that this option can only be used with the "rb", "rbr", and "direct" clustering methods.

-rowmodel=string **vcluster**

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

none The columns of each row are not scaled and used as they are provided in the input file. This is the default setting.

maxtf 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.

sqrt 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.

log 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=string **vcluster**

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

none The columns of the matrix are not globally scaled, and they are used as is. This is the default setting used by vcluster when the correlation coefficient-based similarity function is used.

idf 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 vcluster 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.

The choice of the options for both *-rowmodel* and *-colmodel* were motivated by the clustering requirements of high-dimensional datasets arising in document and commercial datasets. However, for other domains the provided options may not be sufficient. In such domains, the data should be pre-processed to apply the desired row/column model before supplying them to CLUTO. In that case *-rowmodel=none* and *-colmodel=none* should probably be used.

-colprune=float **vcluster**

Selects the factor by which vcluster 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 $p = 0.9$, vcluster 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 vcluster 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.

-nnbrs=int **vcluster & scluster**

This parameter specifies the number of nearest neighbors of each object that will be used in creating the nearest neighbor graph that is used by the graph-partitioning based clustering algorithm. The exact approach of combining these nearest-neighbors to create the graph is controlled by the *-grmodel* parameter. The default value for this parameter is set to 40.

-grmodel=string **vcluster & scluster**

This parameter controls the type of nearest-neighbor graph that will be constructed on the fly and supplied to the graph-partitioning based clustering algorithm. The possible values are:

sd *Symmetric-Direct*
A graph is constructed so that there will be an edge between two objects u and v if and only if both of them are in the nearest-neighbor lists of each other. That is, v is one of the *nnbr*s of u and vice versa. The weight of this edge is set equal to the similarity of the objects (or inversely related to their distance). This is the default option used by both vcluster and scluster.

ad *Asymmetric-Direct*
A graph is constructed so that there will be an edge between two objects u and v as long as one of them is in the nearest-neighbor lists of the other. That is, v is one of the *nnbr*s of u and/or u is one of the *nnbr*s of v. The weight of this edge is set equal to the similarity of the objects (or inversely related to their distance).

sl *Symmetric-Link*
A graph is constructed that has exactly the same adjacency structure as that of the "sd" option. However, the weight of each edge (u, v) is set equal to the number of vertices that are in common in the adjacency lists of u and v (*i.e.*, is equal to the number of shared nearest neighbors). We will refer to this as the $link(u, v)$ count between u and v. This option was motivated by the *link* graph used by the CURE clustering algorithm [1].

al *Asymmetric-Link*
A graph is constructed that has exactly the same adjacency structure as that of the "ad" option. However, the weight of each edge (u, v) is set in a fashion similar to "sl".

none This option is used only by scluster and indicates that the input graph will be used as is.

-edgeprune=float **vcluster & scluster**

This parameter can be used to eliminate certain edges from the nearest-neighbor graph that will tend to connect vertices belonging to different clusters. In particular, if x is the supplied parameter, then an edge (u, v) will be eliminated if and only if

$$link(u, v) < x * nnbrs,$$

where $link(u, v)$ is as defined in *-grmodel=sl*, and *nnbrs* is the number of nearest neighbors used in creating the graph.

The basic motivation behind this pruning method is that if two vertices are part of the same cluster they should be part of a well-connected subgraph (*i.e.*, be part of a sufficiently large clique-like subgraph). Consequently, their adjacency lists must have many common vertices. If that does not happen, then that edge may have been created because these objects matched in non-relevant aspects of their feature vectors, or it may be an edge *bridging* separate clusters. In either case, it can potentially be eliminated.

The default value of this parameter is set to -1, indicating no edge-pruning. Reasonable values for this parameter are within [0.0, 0.5] when *-grmodel* is 'sd' or 'sl', and [1.0, 1.5] when *-grmodel* is 'ad' or 'al'.

Note that this parameter is used only by the graph-partitioning based clustering algorithm.

-vtxprune=float **vcluster & scluster**

This parameter is used to eliminate certain vertices from the nearest-neighbor graph that tend to be outliers. In particular, if x is the supplied parameter, then a vertex u will be eliminated if its degree is less than $x * nnbrs$. The key idea behind this method, especially when the *symmetric* graph models are used, is that if a particular vertex u is not in the the nearest-neighbor list of its nearest-neighbors, then it will most likely be an outlier.

The default value of this parameter is set to -1, indicating no vertex-pruning. Reasonable values for this parameter are within [0.0, 0.5] when *-grmodel* is 'sd' or 'sl', and [1.0, 1.5] when *-grmodel* is 'ad' or 'al'. Note that by using relatively large values for *-edgeprune* and *-vtxprune* you can obtain a graph that contains many small connected components. Such components often correspond to tight clusters in the dataset. This is illustrated in Figure 4. Note that the clustering solution in this example has 48 connected components larger than five vertices, containing only 1345 out of the 8580 objects (please refer to Section 3.2 to find out how to interpret these results).

The vertex-pruning is applied after the edge-pruning has been done.

Note that this parameter is used only by the graph-partitioning based clustering algorithm.

-mincomponent=int **vcluster & scluster**

This parameter is used to eliminate small connected components from the nearest-neighbor graph prior to clustering. In general, if the edge- and vertex-pruning options are used, the resulting graph may have a large number of small connect components (in addition to larger ones). By eliminating (*i.e.*, not clustering) the smaller components eliminates some of the clutter in the resulting clustering solution, and it removes some additional outliers. The default value for this parameter is set to five.

Note that this parameter is used only by the graph-partitioning based clustering algorithm.

-ntrials=int **vcluster & scluster**

Selects the number of different clustering solutions to be computed by the various partitional algorithms. If l is the supplied number, then vcluster and scluster 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 vcluster is 10.

-niter=int **vcluster & scluster**

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. This parameter applies only to the partitional clustering algorithms. The default value is set to 10.

-seed=int **vcluster & scluster**

Selects the seed of the random number generator to be used by vcluster and scluster.

3.1.2 Reporting and Analysis Parameters

There are a total of 14 different optional parameters that control the amount of information that vcluster and scluster report about the clusters, as well as, the analysis that they perform on the discovered clusters. The name and function of these parameters is as follows:

-nooutput **vcluster & scluster**

Specifies that vcluster and scluster should not write the clustering vector and/or agglomerative trees onto the disk.

-clustfile=string **vcluster & scluster**

Specifies the name of the file onto which the clustering vector should be written. The format of this file is described in Section 3.4.1 If this parameter is not specified, then the clustering vector is written to the *MatrixFile.clustering.NClusters* (*GraphFile.clustering.NClusters*) file, where *MatrixFile* (*GraphFile*) is the name of the file that stores the matrix (graph) to be clustered, and *NClusters* is the number of desired clusters.

```
prompt% vcluster -rclassfile=sports.rclass -clmethod=graph -edgeprune=0.4 -vtxprune=0.4 sports.mat 1
*******************************************************************************
vcluster (CLUTO 2.1) Copyright 2001-02, Regents of the University of Minnesota

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

Options ----------------------------------------------------------------------
  CLMethod=GRAPH, CRfun=Cut, SimFun=Cosine, #Clusters: 1
  RowModel=None, ColModel=IDF, GrModel=SY-DIR, NNbrs=40
  Colprune=1.00, EdgePrune=0.40, VtxPrune=0.40, MinComponent=5
  CSType=Best, AggloFrom=0, AggloCRFun=SLINK_W, NTrials=10, NIter=10

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

---------------------------------------------------------------------------------------
48-way clustering: [Cut=7.19e+03] [1345 of 8580], Entropy: 0.086, Purity: 0.929
---------------------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  Entpy Purty | base bask foot hock boxi bicy golf
---------------------------------------------------------------------------------------
  0    41 +0.776 +0.065 +0.000 +0.000 0.000 1.000 |   41    0    0    0    0    0    0
  1    41 +0.745 +0.067 +0.000 +0.000 0.000 1.000 |   41    0    0    0    0    0    0
  2    11 +0.460 +0.059 +0.000 +0.000 0.000 1.000 |    0   11    0    0    0    0    0
  3    11 +0.439 +0.055 +0.000 +0.001 0.157 0.909 |    0    1   10    0    0    0    0
  4    33 +0.426 +0.159 +0.000 +0.000 0.432 0.727 |    3    1   24    5    0    0    0
  5    33 +0.434 +0.119 +0.000 +0.000 0.000 1.000 |    0    0   33    0    0    0    0
  6     9 +0.410 +0.031 +0.001 +0.000 0.000 1.000 |    0    0    9    0    0    0    0
  7    29 +0.400 +0.087 +0.000 +0.000 0.000 1.000 |    0   29    0    0    0    0    0
  8    14 +0.402 +0.058 +0.000 +0.000 0.000 1.000 |   14    0    0    0    0    0    0
  9    21 +0.399 +0.091 +0.000 +0.000 0.000 1.000 |    0    0   21    0    0    0    0
 10    36 +0.381 +0.067 +0.000 +0.000 0.000 1.000 |    0    0    0    0    0   36    0
 11    27 +0.375 +0.050 +0.000 +0.000 0.000 1.000 |    0    0    0   27    0    0    0
 12    41 +0.370 +0.071 +0.000 +0.000 0.000 1.000 |    0   41    0    0    0    0    0
 13    39 +0.371 +0.095 +0.000 +0.000 0.687 0.487 |    7    9   19    2    1    0    1
 14    37 +0.366 +0.088 +0.000 +0.000 0.000 1.000 |    0    0   37    0    0    0    0
 15    18 +0.357 +0.043 +0.000 +0.000 0.000 1.000 |    0   18    0    0    0    0    0
 16    10 +0.351 +0.021 +0.000 +0.000 0.000 1.000 |   10    0    0    0    0    0    0
 17     5 +0.345 +0.012 +0.000 +0.000 0.000 1.000 |    5    0    0    0    0    0    0
 18    23 +0.345 +0.055 +0.000 +0.000 0.000 1.000 |   23    0    0    0    0    0    0
 19    12 +0.340 +0.043 +0.000 +0.000 0.000 1.000 |   12    0    0    0    0    0    0
 20    20 +0.328 +0.059 +0.000 +0.000 0.000 1.000 |    0    0   20    0    0    0    0
 21    18 +0.323 +0.040 +0.001 +0.001 0.000 1.000 |    0    0   18    0    0    0    0
 22     5 +0.316 +0.025 +0.000 +0.000 0.000 1.000 |    5    0    0    0    0    0    0
 23     8 +0.314 +0.021 +0.000 +0.000 0.289 0.750 |    0    2    6    0    0    0    0
 24    12 +0.321 +0.036 +0.000 +0.000 0.000 1.000 |   12    0    0    0    0    0    0
 25    36 +0.312 +0.054 +0.001 +0.001 0.065 0.972 |   35    0    1    0    0    0    0
 26     7 +0.305 +0.040 +0.000 +0.000 0.000 1.000 |    0    0    7    0    0    0    0
 27    25 +0.321 +0.042 +0.000 +0.000 0.000 1.000 |    0   25    0    0    0    0    0
 28    23 +0.309 +0.047 +0.000 +0.000 0.000 1.000 |   23    0    0    0    0    0    0
 29    41 +0.297 +0.056 +0.001 +0.001 0.000 1.000 |   41    0    0    0    0    0    0
 30    20 +0.293 +0.053 +0.000 +0.000 0.000 1.000 |    0   20    0    0    0    0    0
 31    30 +0.294 +0.068 +0.000 +0.000 0.000 1.000 |   30    0    0    0    0    0    0
 32    14 +0.280 +0.032 +0.000 +0.000 0.000 1.000 |    0    0    0    0    0    0   14
 33    37 +0.290 +0.054 +0.000 +0.000 0.000 1.000 |    0    0    0   37    0    0    0
 34    45 +0.273 +0.097 +0.000 +0.000 0.000 1.000 |    0    0    0    0   45    0    0
 35    22 +0.257 +0.046 +0.000 +0.000 0.000 1.000 |    0    0    0    0    0    0   22
 36    36 +0.267 +0.064 +0.000 +0.000 0.406 0.556 |    1   15   20    0    0    0    0
 37    34 +0.251 +0.075 +0.000 +0.000 0.068 0.971 |   33    1    0    0    0    0    0
 38    31 +0.249 +0.065 +0.000 +0.000 0.146 0.935 |    0   29    1    1    0    0    0
 39    36 +0.247 +0.062 +0.000 +0.000 0.000 1.000 |    0   36    0    0    0    0    0
 40    26 +0.255 +0.088 +0.000 +0.000 0.000 1.000 |   26    0    0    0    0    0    0
 41    20 +0.241 +0.046 +0.000 +0.000 0.000 1.000 |    0    0    0    0    0    0   20
 42    26 +0.236 +0.083 +0.000 +0.000 0.000 1.000 |    0   26    0    0    0    0    0
 43     5 +0.297 +0.081 +0.000 +0.000 0.000 1.000 |    0    0    0    5    0    0    0
 44    36 +0.170 +0.053 +0.000 +0.000 0.000 1.000 |    0    0    0    0    0   36    0
 45    84 +0.145 +0.046 +0.000 +0.001 0.000 1.000 |    0    0   84    0    0    0    0
 46    64 +0.147 +0.055 +0.000 +0.001 0.000 1.000 |    0    0   64    0    0    0    0
 47    93 +0.111 +0.047 +0.000 +0.000 0.504 0.527 |   37    2   49    3    2    0    0
---------------------------------------------------------------------------------------

Timing Information -----------------------------------------------------------
   I/O:                                   1.570 sec
   Clustering:                           12.620 sec
   Reporting:                             0.010 sec
*******************************************************************************
```

Figure 4: Output of vcluster for matrix *sports.mat* using 0.4 for edge- and vertex-prune.

-treefile=string **vcluster & scluster**

Specifies the name of the file onto which the hierarchical agglomerative tree should be written. This tree is created either when *-clmethod=agglo*, or when *-fulltree* was specified. The format of this file is described in Section 3.4.2. By default, the tree is written in the file *MatrixFile.tree* (*GraphFile.tree*), where *MatrixFile* (*GraphFile*) is the name of the file storing the input matrix (graph).

-cltreefile=string **vcluster & scluster**

Specifies the name of the file onto which the hierarchical agglomerative tree build on top of the clustering solution should be written. This tree is created either when *-showtree*, was specified. The format of this file is described in Section 3.4.2. By default, the tree is written in the file *MatrixFile.cltree.NClusters* (*GraphFile.cltree.NClusters*) , where *MatrixFile* (*GraphFile*) is the name of the file storing the input matrix (graph), and *NClusters* is the number of desired clusters.

-clabelfile=string **vcluster**

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*, *-showsummaries*, or the *-labeltree* options are specified. The format of this file is described in Section 3.3.4. If this parameter is not specified, vcluster 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 **vcluster & scluster**

Specifies the name of the file that stores the labels of the rows (vertices). The labels of the rows (vertices) are used for reporting purposes when the *-plotmatrix* or the *-plotsmatrix* options are specified. The format of this file is described in Section 3.3.3. If this parameter is not specified, vcluster (scluster) looks to see if a file called *MatrixFile.rlabel* (*GraphFile.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 or vertex becomes "rowj" (*i.e.*, it is labeled by its corresponding row-id).

-rclassfile=string **vcluster & scluster**

Specifies the name of the file that stores the class-labels of the rows (vertices) (*i.e.*, the objects to be clustered). This is used by vcluster (scluster) 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 3.3.5. If this parameter is not specified, vcluster (scluster) looks to see if a file called ***MatrixFile.rlabel*** (*GraphFile.rlabel*) exists, and if it does, reads this file, instead. If no file is provided or the default file does not exist, vcluster and scluster assume that the class labels of the objects are not known and do not perform any cluster-quality analysis based on external measures.

-showfeatures **vcluster**

This parameter instructs vcluster to analyze the discovered clusters and identify the set of features (*i.e.*, columns of the matrix) that are most descriptive of each cluster and 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=none*. This analysis can only be performed when the similarity between objects is computed using the cosine or correlation coefficient.

-showsummaries=string **vcluster**

This parameter instructs vcluster to analyze the discovered clusters and identify relations among the set of most descriptive features of each cluster. The key motivation behind this option is that some of the discovered clusters may contain within them smaller sub-clusters. As a result, by simply looking at the output of *-showfeatures* it may be hard to identify which features go together in these sub-clusters (if they exist). To overcome this problem, *-showsummaries* analyzes the most descriptive features of each cluster and finds subsets of these features that tend to occur together in the objects.

CLUTO provides two different methods for determining which features "go together". These methods are selected by providing the appropriate method-name as an option for this parameter. The possible values are:

cliques Represents the most descriptive features via a graph in which to features are connected via an edge if and only if their co-occurrence frequency within the cluster is greater than their expected co-occurrence. Now given this graph, CLUTO decomposes it into maximal cliques, and uses these cliques as the summaries.

itemsets It mines the objects of each cluster and identifies: (i) maximal frequent itemsets, and (ii) non-maximal itemsets whose support is much higher than that of its maximal supersets. These itemsets are returned as the summaries.

-nfeatures=int **vcluster**

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 **vcluster & scluster**

This parameter instructs vcluster and scluster to build and display 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. The criterion function used in building this tree is controlled by the *-agglocrfun* parameter. If this parameter is not specified then the criterion function used to build the clustering solution is used for all method except *-clmethod=graph*, for which the *wslink* is used.

-labeltree **vcluster & scluster**

This parameter instructs vcluster and scluster 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. This analysis can only be performed when the similarity between objects is computed using the cosine or correlation coefficient.

-zscores **vcluster & scluster**

This parameter instructs vcluster and scluster 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 3.4.1.

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.

-help **vcluster & scluster**

This options instructs vcluster to print a short description of the various command line parameters.

3.1.3 Cluster Visualization Parameters

The vcluster and scluster clustering programs 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 nine optional parameters that control the type of visualization that vcluster performs. The name and function of these parameters is as follows:

-plotformat=string **vcluster & scluster**

Selects the format of the graphics files produced by the visualizations. The possible values for this option are:

- **ps** Outputs an encapsulated postscript[1] 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).
- **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.

-plottree=string **vcluster & scluster**

Produces a graphic representation of the entire hierarchical tree produced when *-clmethod=agglo* or when the *-fulltree* option was specified. The leaves of this tree are labeled based on the supplied row labels (*i.e.*, via the *-rlabelfile* parameter).

-plotmatrix=string **vcluster**

Produces 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

[1] 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.

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.

-plotsmatrix=string **vcluster & scluster**

This visualization is similar to that produced by *-plotmatrix* but was designed to visualize the similarity graph. In this plot, both the rows and columns of the displayed visualization correspond to the vertices of the graph.

-plotclusters=string **vcluster**

Produces a visualization that shows how the clusters are related to each other, 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*.

-plotsclusters=string **vcluster & scluster**

This visualization is similar to that produced by *-plotclusters* but was designed to visualize the similarity between the clusters. In this plot, both the rows and columns of the displayed visualization correspond to the graph clusters.

-clustercolumns **vcluster**

Instructs vcluster 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 **vcluster & scluster**

Instructs vcluster and scluster 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 **vcluster & scluster**

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

3.2 Understanding the Information Produced by CLUTO's Clustering Programs

From the description of vcluster's and scluster'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 vcluster's output, since it is similar to that produced by scluster.

3.2.1 Internal Cluster Quality Statistics

The simpler statistics reported by vcluster & scluster 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 vcluster'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=2.29e+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. In the same line, both programs also display how many of the original objects they were able to cluster (*i.e.*, "`[8204 of 8204]`"). In general, both vcluster and scluster try to cluster all objects. However, when some of the objects (vertices) do not share any dimensions (edges) with the rest of the objects, or when the various edge- and vertex-pruning parameters are used, both programs may end up clustering fewer than the total number of input objects.

After that, vcluster 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 discovered clusters 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.

3.2.2 External Cluster Quality Statistics

In addition to the internal cluster quality measures, vcluster & scluster 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 5 shows the output of vcluster 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 vcluster 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.

```
prompt% vcluster -rclassfile=sports.rclass sports.mat 10
*******************************************************************************
vcluster (CLUTO 2.1) Copyright 2001-02, Regents of the University of Minnesota

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

Options ----------------------------------------------------------------------
  CLMethod=RB, CRfun=I2, SimFun=Cosine, #Clusters: 10
  RowModel=None, ColModel=IDF, GrModel=SY-DIR, NNbrs=40
  Colprune=1.00, EdgePrune=-1.00, VtxPrune=-1.00, MinComponent=5
  CSType=Best, AggloFrom=0, AggloCRFun=I2, NTrials=10, NIter=10

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

--------------------------------------------------------------------------------------
10-way clustering: [I2=2.29e+03] [8580 of 8580], Entropy: 0.155, Purity: 0.885
--------------------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  Entpy Purty | base bask foot hock boxi bicy golf
--------------------------------------------------------------------------------------
  0   359 +0.168 +0.050 +0.020 +0.005 0.010 0.997 |    0  358    1    0    0    0    0
  1   629 +0.106 +0.041 +0.022 +0.007 0.006 0.998 |  628    0    1    0    0    0    0
  2   795 +0.102 +0.036 +0.018 +0.006 0.020 0.995 |    1    1    1  791    0    0    1
  3   762 +0.099 +0.034 +0.021 +0.006 0.010 0.997 |    0    1  760    0    0    0    1
  4   482 +0.098 +0.045 +0.022 +0.009 0.015 0.996 |    0  480    1    1    0    0    0
  5   844 +0.095 +0.035 +0.023 +0.007 0.023 0.993 |  838    0    5    0    1    0    0
  6  1724 +0.059 +0.026 +0.022 +0.007 0.016 0.996 | 1717    3    3    1    0    0    0
  7  1175 +0.051 +0.015 +0.021 +0.006 0.024 0.992 |    8    1 1166    0    0    0    0
  8   853 +0.043 +0.015 +0.019 +0.006 0.461 0.619 |   46  528  265    8    0    0    6
  9   957 +0.032 +0.012 +0.015 +0.006 0.862 0.343 |  174   38  143    8  121  145  328
--------------------------------------------------------------------------------------

Timing Information -----------------------------------------------------------
   I/O:                                   1.620 sec
   Clustering:                            9.110 sec
   Reporting:                             0.230 sec
*******************************************************************************
```

Figure 5: Output of vcluster for matrix *sports.mat* and a 10-way clustering that uses external quality measures.

Looking at Figure 5 we can see that vcluster, 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 [6]. 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 360 documents about basketball, and two documents about football. Looking at this class-distribution table, we can easily determine the quality of the different clusters.

3.2.3 Looking at each Cluster's Features

By specifying the *-showfeatures* option, vcluster 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 6 shows the output produced by vcluster 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, vcluster 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, vcluster 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 "warrior" explains 38.4% 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, vcluster outputs the column number of each feature instead of its label. This is illustrated in Figure 7 for the same problem in which *-clabelfile* was not specified. Note that the columns are numbered from one.

By specifying the *-showsummaries* option, vcluster will further analyze the most descriptive features of each cluster and try to identify the set of features that co-occur in the objects. Figure 8 shows the output produced by vcluster when *-showsummaries=cliques* was specified and when a file was provided with the labels of each one of the columns (via the *-clabelfile* option). Note that some clusters contain only a single summary; however, many clusters have more than one summary associated with them. In many cases there is a large overlap between the features of the various summaries of the same cluster, but the unique features of each summary does provide some clues on particular subsets of objects within each cluster.

3.2.4 Looking at the Hierarchical Agglomerative Tree

The vcluster & scluster 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 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 9. This result was obtained by

```
prompt% vcluster -rclassfile=sports.rclass -clabelfile=sports.clabel -showfeatures sports.mat 10
*******************************************************************************
vcluster (CLUTO 2.1) Copyright 2001-02, Regents of the University of Minnesota

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

Options ----------------------------------------------------------------------
  CLMethod=RB, CRfun=I2, SimFun=Cosine, #Clusters: 10
  RowModel=None, ColModel=IDF, GrModel=SY-DIR, NNbrs=40
  Colprune=1.00, EdgePrune=-1.00, VtxPrune=-1.00, MinComponent=5
  CSType=Best, AggloFrom=0, AggloCRFun=I2, NTrials=10, NIter=10

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

---------------------------------------------------------------------------------------
10-way clustering: [I2=2.29e+03] [8580 of 8580], Entropy: 0.155, Purity: 0.885
---------------------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  Entpy Purty | base bask foot hock boxi bicy golf
---------------------------------------------------------------------------------------
  0   359 +0.168 +0.050 +0.020 +0.005 0.010 0.997 |    0  358    1    0    0    0    0
  1   629 +0.106 +0.041 +0.022 +0.007 0.006 0.998 |  628    0    1    0    0    0    0
  2   795 +0.102 +0.036 +0.018 +0.006 0.020 0.995 |    1    1    1  791    0    0    1
  3   762 +0.099 +0.034 +0.021 +0.006 0.010 0.997 |    0    1  760    0    0    0    1
  4   482 +0.098 +0.045 +0.022 +0.009 0.015 0.996 |    0  480    1    1    0    0    0
  5   844 +0.095 +0.035 +0.023 +0.007 0.023 0.993 |  838    0    5    0    1    0    0
  6  1724 +0.059 +0.026 +0.022 +0.007 0.016 0.996 | 1717    3    3    1    0    0    0
  7  1175 +0.051 +0.015 +0.021 +0.006 0.024 0.992 |    8    1 1166    0    0    0    0
  8   853 +0.043 +0.015 +0.019 +0.006 0.461 0.619 |   46  528  265    8    0    0    6
  9   957 +0.032 +0.012 +0.015 +0.006 0.862 0.343 |  174   38  143    8  121  145  328
---------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
10-way clustering solution - Descriptive & Discriminating Features...
--------------------------------------------------------------------------------
Cluster   0, Size:   359, ISim: 0.168, ESim: 0.020
      Descriptive:  warrior 38.1%, hardawai  6.9%, mullin  6.1%, nelson  4.4%, richmond  4.2%
   Discriminating:  warrior 26.6%, hardawai  4.9%, mullin  4.3%, richmond  2.9%, g  2.7%

Cluster   1, Size:   629, ISim: 0.106, ESim: 0.022
      Descriptive:  canseco  9.0%, henderson  7.5%, russa  6.3%, la  3.8%, mcgwire  3.2%
   Discriminating:  canseco  7.5%, henderson  5.9%, russa  5.3%, la  2.6%, mcgwire  2.6%

Cluster   2, Size:   795, ISim: 0.102, ESim: 0.018
      Descriptive:  shark 22.3%, goal  9.4%, nhl  4.4%, period  3.4%, penguin  1.6%
   Discriminating:  shark 17.1%, goal  5.9%, nhl  3.4%, period  2.3%, giant  1.5%

Cluster   3, Size:   762, ISim: 0.099, ESim: 0.021
      Descriptive:  yard 35.8%, pass  7.7%, touchdown  6.5%, td  2.6%, kick  2.1%
   Discriminating:  yard 28.2%, pass  5.4%, touchdown  5.1%, td  2.1%, kick  1.5%

Cluster   4, Size:   482, ISim: 0.098, ESim: 0.022
      Descriptive:  laker  6.0%, nba  3.4%, bull  3.0%, rebound  2.9%, piston  2.5%
   Discriminating:  laker  4.9%, nba  2.7%, bull  2.5%, piston  2.2%, jammer  2.1%

Cluster   5, Size:   844, ISim: 0.095, ESim: 0.023
      Descriptive:  giant 20.7%, mitchell  4.8%, craig  3.3%, mcgee  2.4%, clark  2.0%
   Discriminating:  giant 15.6%, mitchell  4.3%, craig  2.5%, mcgee  2.2%, yard  1.9%

Cluster   6, Size:  1724, ISim: 0.059, ESim: 0.022
      Descriptive:  in  5.6%, hit  5.2%, homer  2.6%, run  2.4%, sox  2.2%
   Discriminating:  in  4.1%, hit  3.4%, yard  2.8%, sox  2.1%, homer  1.8%

Cluster   7, Size:  1175, ISim: 0.051, ESim: 0.021
      Descriptive:  seifert  3.2%, bowl  3.2%, montana  3.1%, raider  2.5%, super  2.0%
   Discriminating:  seifert  3.6%, montana  3.3%, bowl  3.0%, raider  2.5%, super  2.2%

Cluster   8, Size:   853, ISim: 0.043, ESim: 0.019
      Descriptive:  confer  2.4%, school  2.3%, santa  2.1%, st  1.8%, coach  1.8%
   Discriminating:  giant  2.1%, school  1.9%, confer  1.9%, santa  1.7%, yard  1.5%

Cluster   9, Size:   957, ISim: 0.032, ESim: 0.015
      Descriptive:  box 12.4%, golf  3.9%, hole  2.9%, round  2.4%, par  2.0%
   Discriminating:  box  7.6%, golf  3.7%, hole  2.6%, par  1.9%, round  1.5%
--------------------------------------------------------------------------------

Timing Information -----------------------------------------------------------
   I/O:                                   1.500 sec
   Clustering:                            9.240 sec
   Reporting:                             0.770 sec
*******************************************************************************
```

Figure 6: Output of vcluster for matrix *sports.mat* and a 10-way clustering that shows the descriptive and discriminating features of each cluster.

```
prompt% vcluster -rclassfile=sports.rclass -showfeatures sports.mat 10
*******************************************************************************
vcluster (CLUTO 2.1) Copyright 2001-02, Regents of the University of Minnesota

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

Options ----------------------------------------------------------------------
  CLMethod=RB, CRfun=I2, SimFun=Cosine, #Clusters: 10
  RowModel=None, ColModel=IDF, GrModel=SY-DIR, NNbrs=40
  Colprune=1.00, EdgePrune=-1.00, VtxPrune=-1.00, MinComponent=5
  CSType=Best, AggloFrom=0, AggloCRFun=I2, NTrials=10, NIter=10

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

---------------------------------------------------------------------------------------
10-way clustering: [I2=2.29e+03] [8580 of 8580], Entropy: 0.155, Purity: 0.885
---------------------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  Entpy Purty | base bask foot hock boxi bicy golf
---------------------------------------------------------------------------------------
  0   359 +0.168 +0.050 +0.020 +0.005 0.010 0.997 |    0  358    1    0    0    0    0
  1   629 +0.106 +0.041 +0.022 +0.007 0.006 0.998 |  628    0    1    0    0    0    0
  2   795 +0.102 +0.036 +0.018 +0.006 0.020 0.995 |    1    1    1  791    0    0    1
  3   762 +0.099 +0.034 +0.021 +0.006 0.010 0.997 |    0    1  760    0    0    0    1
  4   482 +0.098 +0.045 +0.022 +0.009 0.015 0.996 |    0  480    1    1    0    0    0
  5   844 +0.095 +0.035 +0.023 +0.007 0.023 0.993 |  838    0    5    0    1    0    0
  6  1724 +0.059 +0.026 +0.022 +0.007 0.016 0.996 | 1717    3    3    1    0    0    0
  7  1175 +0.051 +0.015 +0.021 +0.006 0.024 0.992 |    8    1 1166    0    0    0    0
  8   853 +0.043 +0.015 +0.019 +0.006 0.461 0.619 |   46  528  265    8    0    0    6
  9   957 +0.032 +0.012 +0.015 +0.006 0.862 0.343 |  174   38  143    8  121  145  328
---------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
10-way clustering solution - Descriptive & Discriminating Features...
--------------------------------------------------------------------------------
Cluster   0, Size:   359, ISim: 0.168, ESim: 0.020
      Descriptive:  col02843 38.1%, col06054  6.9%, col03655  6.1%, col01209  4.4%, col11248  4.2%
   Discriminating:  col02843 26.6%, col06054  4.9%, col03655  4.3%, col11248  2.9%, col20475  2.7%

Cluster   1, Size:   629, ISim: 0.106, ESim: 0.022
      Descriptive:  col18174  9.0%, col11733  7.5%, col18183  6.3%, col01570  3.8%, col26743  3.2%
   Discriminating:  col18174  7.5%, col11733  5.9%, col18183  5.3%, col01570  2.6%, col26743  2.6%

Cluster   2, Size:   795, ISim: 0.102, ESim: 0.018
      Descriptive:  col04688 22.3%, col00134  9.4%, col04423  4.4%, col02099  3.4%, col04483  1.6%
   Discriminating:  col04688 17.1%, col00134  5.9%, col04423  3.4%, col02099  2.3%, col01536  1.5%

Cluster   3, Size:   762, ISim: 0.099, ESim: 0.021
      Descriptive:  col00086 35.8%, col00091  7.7%, col00084  6.5%, col01091  2.6%, col00132  2.1%
   Discriminating:  col00086 28.2%, col00091  5.4%, col00084  5.1%, col01091  2.1%, col00132  1.5%

Cluster   4, Size:   482, ISim: 0.098, ESim: 0.022
      Descriptive:  col10737  6.0%, col03412  3.4%, col00597  3.0%, col00541  2.9%, col06527  2.5%
   Discriminating:  col10737  4.9%, col03412  2.7%, col00597  2.5%, col06527  2.2%, col51202  2.1%

Cluster   5, Size:   844, ISim: 0.095, ESim: 0.023
      Descriptive:  col01536 20.7%, col04716  4.8%, col04640  3.3%, col03838  2.4%, col01045  2.0%
   Discriminating:  col01536 15.6%, col04716  4.3%, col04640  2.5%, col03838  2.2%, col00086  1.9%

Cluster   6, Size:  1724, ISim: 0.059, ESim: 0.022
      Descriptive:  col04265  5.6%, col00281  5.2%, col13856  2.6%, col00340  2.4%, col01362  2.2%
   Discriminating:  col04265  4.1%, col00281  3.4%, col00086  2.8%, col01362  2.1%, col13856  1.8%

Cluster   7, Size:  1175, ISim: 0.051, ESim: 0.021
      Descriptive:  col02393  3.2%, col00024  3.2%, col10761  3.1%, col00031  2.5%, col00147  2.0%
   Discriminating:  col02393  3.6%, col10761  3.3%, col00024  3.0%, col00031  2.5%, col00147  2.2%

Cluster   8, Size:   853, ISim: 0.043, ESim: 0.019
      Descriptive:  col00910  2.4%, col00616  2.3%, col01186  2.1%, col00428  1.8%, col00057  1.8%
   Discriminating:  col01536  2.1%, col00616  1.9%, col00910  1.9%, col01186  1.7%, col00086  1.5%

Cluster   9, Size:   957, ISim: 0.032, ESim: 0.015
      Descriptive:  col00351 12.4%, col01953  3.9%, col00396  2.9%, col00532  2.4%, col16968  2.0%
   Discriminating:  col00351  7.6%, col01953  3.7%, col00396  2.6%, col16968  1.9%, col00532  1.5%
--------------------------------------------------------------------------------

Timing Information -----------------------------------------------------------
   I/O:                                 1.530 sec
   Clustering:                          9.070 sec
   Reporting:                           0.730 sec
*******************************************************************************
```

Figure 7: Output of vcluster for matrix *sports.mat* and a 10-way clustering that shows the descriptive and discriminating features of each cluster.

```
prompt% vcluster -rclassfile=sports.rclass -clabelfile=sports.clabel -nfeatures=8
                 -showsummaries=cliques sports.mat 10
*******************************************************************************
vcluster (CLUTO 2.1) Copyright 2001-02, Regents of the University of Minnesota

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

Options ----------------------------------------------------------------------
  CLMethod=RB, CRfun=I2, SimFun=Cosine, #Clusters: 10
  RowModel=None, ColModel=IDF, GrModel=SY-DIR, NNbrs=40
  Colprune=1.00, EdgePrune=-1.00, VtxPrune=-1.00, MinComponent=5
  CSType=Best, AggloFrom=0, AggloCRFun=I2, NTrials=10, NIter=10

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

---------------------------------------------------------------------------------------
10-way clustering: [I2=2.29e+03] [8580 of 8580], Entropy: 0.155, Purity: 0.885
---------------------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  Entpy Purty | base bask foot hock boxi bicy golf
---------------------------------------------------------------------------------------
  0   359 +0.168 +0.050 +0.020 +0.005 0.010 0.997 |    0  358    1    0    0    0    0
  1   629 +0.106 +0.041 +0.022 +0.007 0.006 0.998 |  628    0    1    0    0    0    0
  2   795 +0.102 +0.036 +0.018 +0.006 0.020 0.995 |    1    1    1  791    0    0    1
  3   762 +0.099 +0.034 +0.021 +0.006 0.010 0.997 |    0    1  760    0    0    0    1
  4   482 +0.098 +0.045 +0.022 +0.009 0.015 0.996 |    0  480    1    1    0    0    0
  5   844 +0.095 +0.035 +0.023 +0.007 0.023 0.993 |  838    0    5    0    1    0    0
  6  1724 +0.059 +0.026 +0.022 +0.007 0.016 0.996 | 1717    3    3    1    0    0    0
  7  1175 +0.051 +0.015 +0.021 +0.006 0.024 0.992 |    8    1 1166    0    0    0    0
  8   853 +0.043 +0.015 +0.019 +0.006 0.461 0.619 |   46  528  265    8    0    0    6
  9   957 +0.032 +0.012 +0.015 +0.006 0.862 0.343 |  174   38  143    8  121  145  328
---------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
10-way clustering solution - Cluster Summaries using Cliques...
--------------------------------------------------------------------------------
Cluster   0, Size:   359, ISim: 0.168, ESim: 0.020
  55.05%    warrior hardawai mullin nelson richmond g marciulioni higgin

Cluster   1, Size:   629, ISim: 0.106, ESim: 0.022
  51.13%    canseco henderson russa la mcgwire gallego steinbach stewart

Cluster   2, Size:   795, ISim: 0.102, ESim: 0.018
  48.03%    goal nhl period penguin wing goali
  51.34%    shark goal nhl wing goali hockei

Cluster   3, Size:   762, ISim: 0.099, ESim: 0.021
  65.34%    yard pass touchdown td kick rush quarter intercept

Cluster   4, Size:   482, ISim: 0.098, ESim: 0.022
  41.63%    rebound score jammer
  45.79%    laker nba bull rebound piston johnson score

Cluster   5, Size:   844, ISim: 0.095, ESim: 0.023
  54.13%    giant mitchell craig mcgee clark in thompson pitch

Cluster   6, Size:  1724, ISim: 0.059, ESim: 0.022
  45.71%    in hit homer run brave red twin
  46.54%    in hit homer run sox red twin

Cluster   7, Size:  1175, ISim: 0.051, ESim: 0.021
  33.24%    bowl montana raider super nfl quarterback lott
  31.90%    seifert montana raider nfl quarterback lott

Cluster   8, Size:   853, ISim: 0.043, ESim: 0.019
  44.20%    santa coach tournam score basketbal
  46.51%    school santa coach basketbal
  46.16%    school santa st coach
  42.81%    confer santa st coach tournam score

Cluster   9, Size:   957, ISim: 0.032, ESim: 0.015
  27.03%    golf hole round par cours tour
  24.35%    box round tyson
--------------------------------------------------------------------------------

Timing Information -----------------------------------------------------------
   I/O:                                 1.520 sec
   Clustering:                          9.340 sec
   Reporting:                           0.740 sec
*******************************************************************************
```

Figure 8: Output of vcluster for matrix *sports.mat* and a 10-way clustering that shows the summaries using maximal cliques.

specifying both *-showtree* as well as the *-rclassfile* parameter that provides the class labels for each object in the matrix.

```
prompt% vcluster -rclassfile=sports.rclass -showtree sports.mat 10
*******************************************************************************
vcluster (CLUTO 2.1) Copyright 2001-02, Regents of the University of Minnesota

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

Options ----------------------------------------------------------------------
  CLMethod=RB, CRfun=I2, SimFun=Cosine, #Clusters: 10
  RowModel=None, ColModel=IDF, GrModel=SY-DIR, NNbrs=40
  Colprune=1.00, EdgePrune=-1.00, VtxPrune=-1.00, MinComponent=5
  CSType=Best, AggloFrom=0, AggloCRFun=I2, NTrials=10, NIter=10

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

---------------------------------------------------------------------------------------
10-way clustering: [I2=2.29e+03] [8580 of 8580], Entropy: 0.155, Purity: 0.885
---------------------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  Entpy Purty | base bask foot hock boxi bicy golf
---------------------------------------------------------------------------------------
  0   359 +0.168 +0.050 +0.020 +0.005 0.010 0.997 |    0  358    1    0    0    0    0
  1   629 +0.106 +0.041 +0.022 +0.007 0.006 0.998 |  628    0    1    0    0    0    0
  2   795 +0.102 +0.036 +0.018 +0.006 0.020 0.995 |    1    1    1  791    0    0    1
  3   762 +0.099 +0.034 +0.021 +0.006 0.010 0.997 |    0    1  760    0    0    0    1
  4   482 +0.098 +0.045 +0.022 +0.009 0.015 0.996 |    0  480    1    1    0    0    0
  5   844 +0.095 +0.035 +0.023 +0.007 0.023 0.993 |  838    0    5    0    1    0    0
  6  1724 +0.059 +0.026 +0.022 +0.007 0.016 0.996 | 1717    3    3    1    0    0    0
  7  1175 +0.051 +0.015 +0.021 +0.006 0.024 0.992 |    8    1 1166    0    0    0    0
  8   853 +0.043 +0.015 +0.019 +0.006 0.461 0.619 |   46  528  265    8    0    0    6
  9   957 +0.032 +0.012 +0.015 +0.006 0.862 0.343 |  174   38  143    8  121  145  328
---------------------------------------------------------------------------------------

-------------------------------------------------------------------------------
Hierarchical Tree that optimizes the I2 criterion function...
-------------------------------------------------------------------------------
                          base bask foot hock boxi bicy golf
----------------------------------------------------------------
18
|-----15
|     |-------------6     1717    3    3    1    0    0    0
|     |---13
|         |---------1      628    0    1    0    0    0    0
|         |---------5      838    0    5    0    1    0    0
|-17
  |---------12
  |         |-------7        8    1 1166    0    0    0    0
  |         |-------3        0    1  760    0    0    0    1
  |-16
    |---14
    |   |-----11
    |   |     |-----8       46  528  265    8    0    0    6
    |   |     |-----9      174   38  143    8  121  145  328
    |   |-------10
    |           |---0        0  358    1    0    0    0    0
    |           |---4        0  480    1    1    0    0    0
    |---------------2        1    1    1  791    0    0    1
----------------------------------------------------------------
-------------------------------------------------------------------------------

Timing Information -----------------------------------------------------------
   I/O:                                  1.490 sec
   Clustering:                           9.310 sec
   Reporting:                            0.660 sec
*******************************************************************************
```

Figure 9: Output of vcluster 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 vcluster 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 vcluster. 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 9 the node numbered 10 represents the first pair of clusters (9 and 7) that were merged, the node numbered 11 represents the second pair of clusters (0 and 5) that were merged, and so on.

In addition to the tree itself, vcluster 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% vcluster -rclassfile=sports.rclass -clabelfile=sports.clabel -showtree -labeltree sports.mat 10
*******************************************************************************
vcluster (CLUTO 2.1) Copyright 2001-02, Regents of the University of Minnesota

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

Options ----------------------------------------------------------------------
  CLMethod=RB, CRfun=I2, SimFun=Cosine, #Clusters: 10
  RowModel=None, ColModel=IDF, GrModel=SY-DIR, NNbrs=40
  Colprune=1.00, EdgePrune=-1.00, VtxPrune=-1.00, MinComponent=5
  CSType=Best, AggloFrom=0, AggloCRFun=I2, NTrials=10, NIter=10

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

---------------------------------------------------------------------------------------
10-way clustering: [I2=2.29e+03] [8580 of 8580], Entropy: 0.155, Purity: 0.885
---------------------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  Entpy Purty | base bask foot hock boxi bicy golf
---------------------------------------------------------------------------------------
  0   359 +0.168 +0.050 +0.020 +0.005 0.010 0.997 |    0  358    1    0    0    0    0
  1   629 +0.106 +0.041 +0.022 +0.007 0.006 0.998 |  628    0    1    0    0    0    0
  2   795 +0.102 +0.036 +0.018 +0.006 0.020 0.995 |    1    1    1  791    0    0    1
  3   762 +0.099 +0.034 +0.021 +0.006 0.010 0.997 |    0    1  760    0    0    0    1
  4   482 +0.098 +0.045 +0.022 +0.009 0.015 0.996 |    0  480    1    1    0    0    0
  5   844 +0.095 +0.035 +0.023 +0.007 0.023 0.993 |  838    0    5    0    1    0    0
  6  1724 +0.059 +0.026 +0.022 +0.007 0.016 0.996 | 1717    3    3    1    0    0    0
  7  1175 +0.051 +0.015 +0.021 +0.006 0.024 0.992 |    8    1 1166    0    0    0    0
  8   853 +0.043 +0.015 +0.019 +0.006 0.461 0.619 |   46  528  265    8    0    0    6
  9   957 +0.032 +0.012 +0.015 +0.006 0.862 0.343 |  174   38  143    8  121  145  328
---------------------------------------------------------------------------------------

------------------------------------------------------------------------------
Hierarchical Tree that optimizes the I2 criterion function...
------------------------------------------------------------------------------
                            Size   ISim      XSim       Gain
18                        [ 8580, 2.57e-02, 0.00e+00, -2.30e+02] [giant  1.7%, yard  1.6%, hit  1.3%, box  1.2%, ]
|-----15                  [ 3197, 4.95e-02, 1.71e-02, -9.17e+01] [in  4.4%, giant  3.7%, hit  3.6%, pitch  2.4%, ]
|     |-------------6     [ 1724, 5.91e-02, 3.60e-02, +0.00e+00] [in  5.6%, hit  5.2%, homer  2.6%, run  2.4%, ]
|     |---13              [ 1473, 6.80e-02, 3.60e-02, -8.10e+01] [giant  9.8%, canseco  2.6%, pitch  2.4%, mitchell  2.3%, ]
|         |---------1     [  629, 1.06e-01, 3.56e-02, +0.00e+00] [canseco  9.0%, henderson  7.5%, russa  6.3%, la  3.8%, ]
|         |---------5     [  844, 9.54e-02, 3.56e-02, +0.00e+00] [giant 20.7%, mitchell  4.8%, craig  3.3%, mcgee  2.4%, ]
|-17                      [ 5383, 2.76e-02, 1.71e-02, -1.35e+02] [yard  3.8%, shark  1.8%, box  1.8%, goal  1.6%, ]
  |---------12            [ 1937, 5.16e-02, 1.94e-02, -6.48e+01] [yard 14.4%, pass  3.8%, touchdown  2.8%, bowl  1.9%, ]
  |         |-------7     [ 1175, 5.09e-02, 3.68e-02, +0.00e+00] [seifert  3.2%, bowl  3.2%, montana  3.1%, raider  2.5%, ]
  |         |-------3     [  762, 9.91e-02, 3.68e-02, +0.00e+00] [yard 35.8%, pass  7.7%, touchdown  6.5%, td  2.6%, ]
  |-16                    [ 3446, 2.93e-02, 1.94e-02, -1.19e+02] [shark  4.2%, warrior  2.9%, goal  2.3%, score  1.8%, ]
    |---14                [ 2651, 2.95e-02, 1.82e-02, -9.06e+01] [warrior  4.9%, box  2.5%, shot  1.4%, score  1.4%, ]
    |   |-----11          [ 1810, 2.70e-02, 1.87e-02, -5.28e+01] [box  5.1%, tournam  2.1%, golf  1.4%, school  1.3%, ]
    |   |     |-----8     [  853, 4.33e-02, 1.66e-02, +0.00e+00] [confer  2.4%, school  2.3%, santa  2.1%, st  1.8%, ]
    |   |     |-----9     [  957, 3.25e-02, 1.66e-02, +0.00e+00] [box 12.4%, golf  3.9%, hole  2.9%, round  2.4%, ]
    |   |-------10        [  841, 8.73e-02, 1.87e-02, -4.98e+01] [warrior 15.3%, laker  4.3%, hardawai  2.6%, mullin  2.3%, ]
    |           |---0     [  359, 1.68e-01, 4.98e-02, +0.00e+00] [warrior 38.1%, hardawai  6.9%, mullin  6.1%, nelson  4.4%, ]
    |           |---4     [  482, 9.84e-02, 4.98e-02, +0.00e+00] [laker  6.0%, nba  3.4%, bull  3.0%, rebound  2.9%, ]
    |---------------2     [  795, 1.02e-01, 1.82e-02, +0.00e+00] [shark 22.3%, goal  9.4%, nhl  4.4%, period  3.4%, ]
------------------------------------------------------------------------------

Timing Information -----------------------------------------------------------
   I/O:                                   1.470 sec
   Clustering:                            9.110 sec
   Reporting:                             1.080 sec
*******************************************************************************
```

Figure 10: Output of vcluster 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, vcluster 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 vcluster in this case is shown in Figure 10.

Looking at this figure we can see that in addition to the tree itself, vcluster 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, 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, and the "Gain" which is the change in the value of the particular clustering criterion function as a result of combining the two child clusters. For example, the cluster corresponding to node 13, contains 1473 documents, whose average similarity is 6.80e-02, the average similarity between the documents in this cluster and the documents in the cluster corresponding to node 10 is 3.60e-02, and as the result of this merging, the value of the criterion function (*i.e.*, $\mathcal{I}_2$ in this example) was decreased by 8.10e+01. Note that since in case of $\mathcal{I}_2$ the goal is to maximize its value, the fact that the gain is negative means that with respect to the criterion function the resulting clustering solution is worse (which was expected).

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.

3.2.5 Looking at the Visualizations

As discussed in Section 3.1 both vcluster and scluster 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 11 shows the type of visualizations that can be produced when *-plotmatrix* is specified for a sparse matrix. In particular, Figure 11(a) shows the visualization produced by executing the following command:

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

As we can see from that plot, vcluster 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 11(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:

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

As we can see from this plot, vcluster 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, vcluster draws the actual hierarchical tree that was computed. Finally, Figure 11(c) shows a visualization of the 10-way clustering solution obtained by scluster. In particular, this plot was obtained by executing the following command:

```
vcluster -clmethod=agglo -clustercolumns -plotmatrix=fig3.ps tr23.mat 10.
```

Figure 12 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:

```
vcluster -sim=corr -plotmatrix=fig4.ps genes1.mat 5
vcluster -sim=corr -fulltree -clustercolumns -plotmatrix=fig5.ps genes1.mat 5
vcluster -sim=corr -clmethod=agglo -clustercolumns -plotmatrix=fig6.ps genes1.mat 5
```

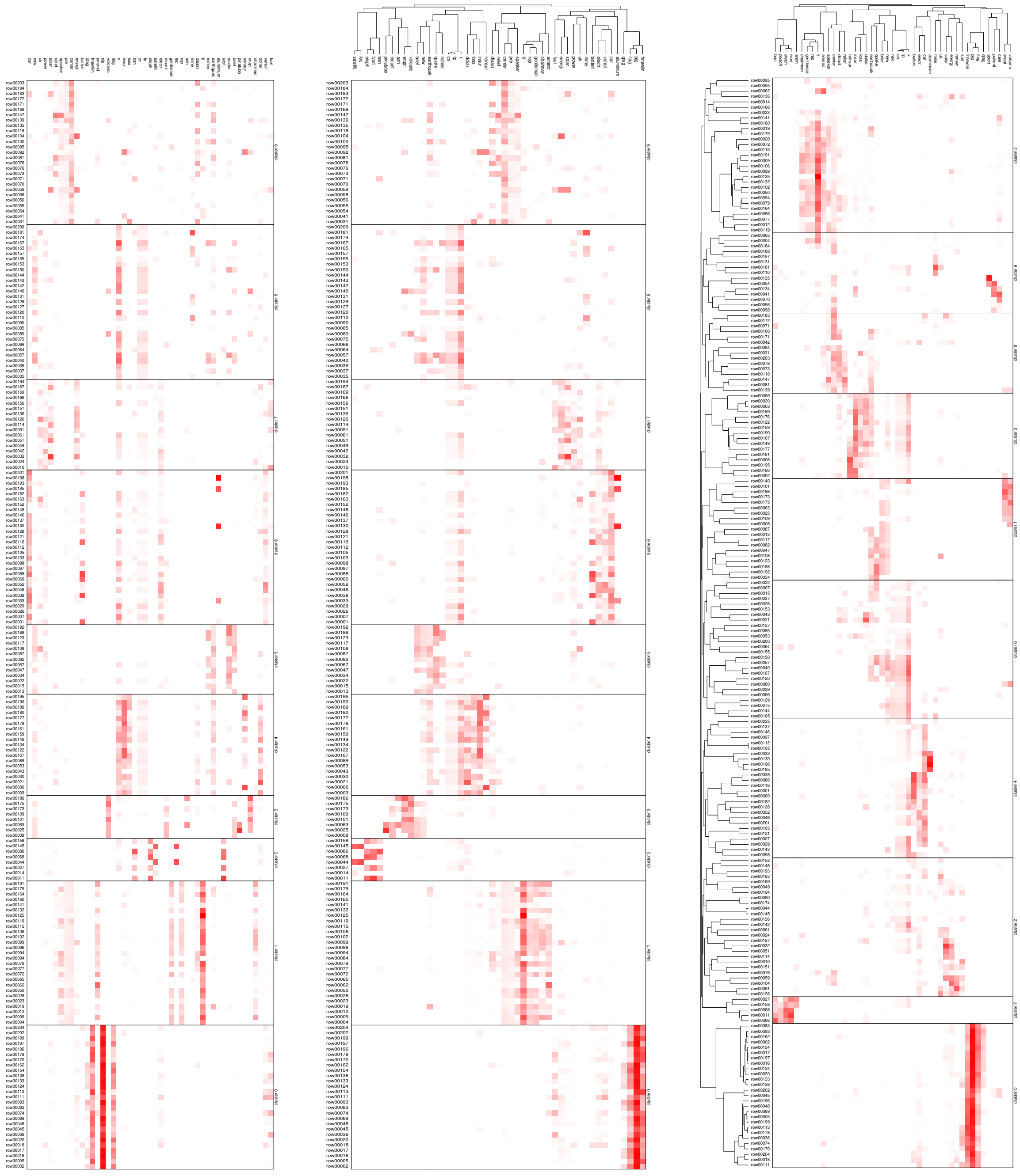
These plots are similar in nature to those produced for sparse matrices and the only difference is that they show all the columns (and not just the union of the descriptive and discriminating features). Also note that each row now has a label (corresponding to the name of the particular gene) that is 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 13 for another micro-array gene expression data set. The different visualizations were produced by executing the following commands:

```
vcluster -clmethod=agglo -plotmatrix=fig7.ps genes2.mat 1
vcluster -clmethod=agglo -zeroblack -plotmatrix=fig8.ps genes2.mat 1
```

Figure 14 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:

```
vcluster -clustercolumns -plotclusters=fig9.ps tr23.mat 10
vcluster -clustercolumns -plotclusters=fig10.ps -nfeatures=10 sports.mat 20
```



(a) (b) (c)

Figure 11: Various visualizations generated by the *-plotmatrix* parameter. (a) Shows the clustering solution produced by vcluster; (b) Shows the same clustering solution but the rows and columns have been re-ordered. (c) Shows the clustering solution produced by scluster.

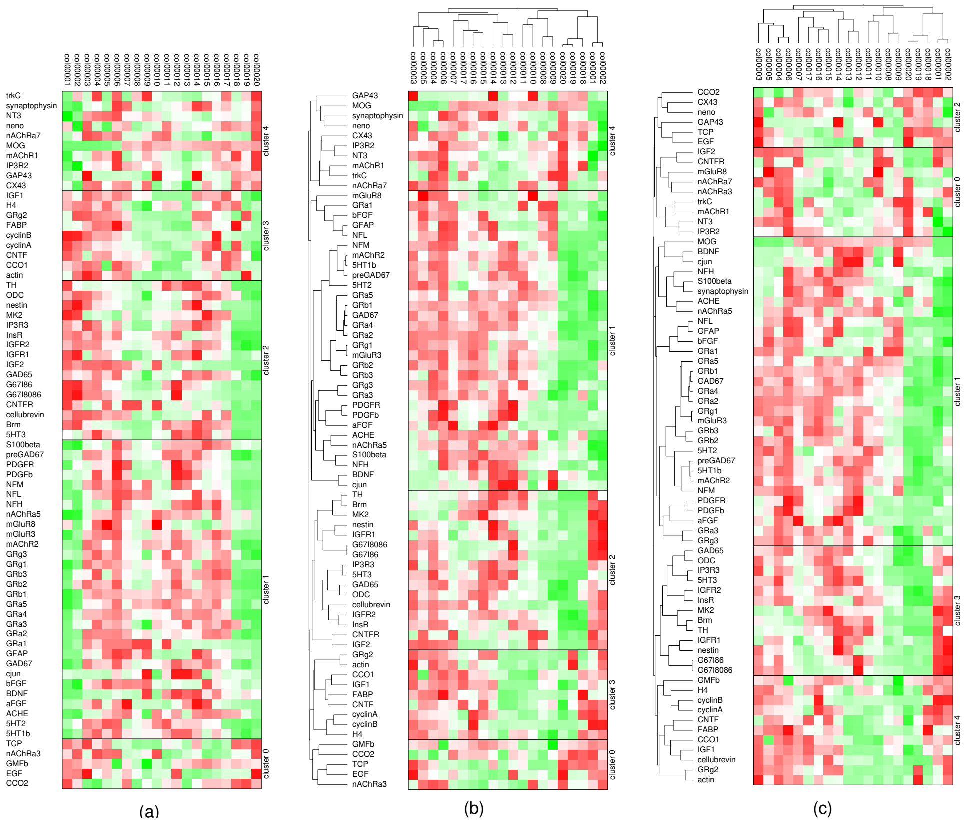


Figure 12: Various visualizations generated by the *-plotmatrix* parameter. (a) Shows the clustering solution produced by the "rb" method of vcluster; (b) Shows the same clustering solution but the rows and columns have been re-ordered. (c) Shows the clustering solution produced by the agglomerative method for vcluster.

This plot shows the clustering solution shown at Figure 11(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 15 shows the type of visualization that can be produced when *-plottree* is specified. This plot was obtained by executing the following command:

```
vcluster -clmethod=agglo -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.

3.3 Input File Formats

The vcluster and scluster programs require an input file that stores the objects to be clustered in a matrix or graph 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.

3.3.1 Matrix File

The primary input of CLUTO's vcluster program 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 the second format is suitable for storing dense matrices. Note that CLUTO, automatically detects the format of the input

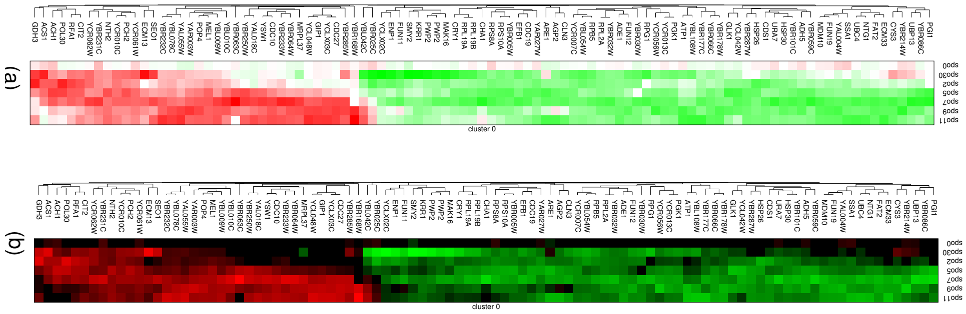


Figure 13: Various visualizations generated by the *-plotmatrix* parameter. (a) Shows the clustering solution produced by the agglomerative method of vcluster; (b) Shows the same clustering solution but the color scheme has been changed.

file based on the first line of the file (*i.e.*, the sparse matrix format has three numbers whereas the dense matrix format has two numbers).

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 16. This figure shows an example 7×8 matrix and its corresponding representation in CLUTO's matrix format.

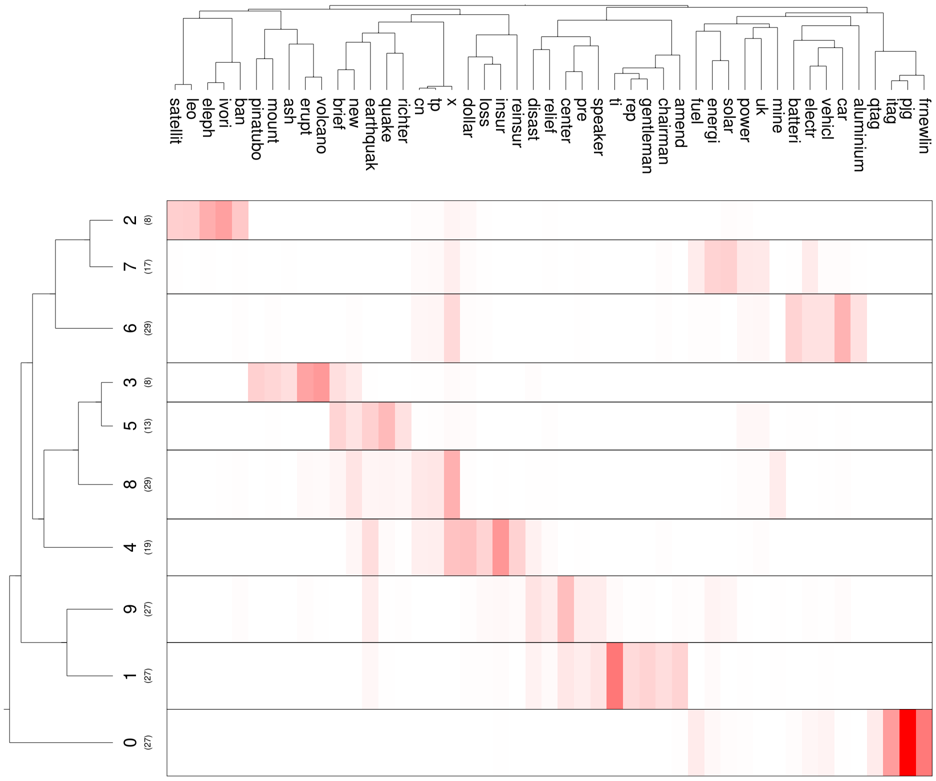
Dense Matrix Format A dense matrix A with n rows and m columns is stored in a plain text file that contains $n+1$ lines. The first line stores information about the size of the matrix, while the remaining n lines contain information for each row of A. The first line of the matrix file contains exactly 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.

3.3.2 Graph File

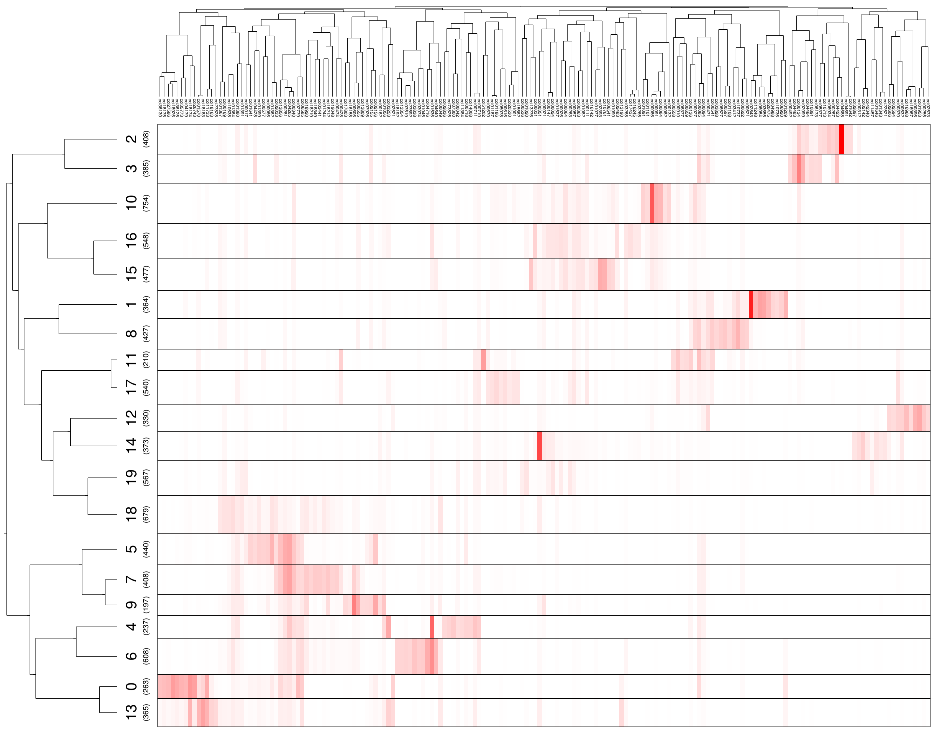
The primary input of CLUTO's scluster program is the adjacency matrix of the graph that specifies the similarity between the objects to be clustered. Each row/column of this matrix represents a single object, and a value at the (i, j) location of this matrix indicates the similarity between the ith and the jth object.

CLUTO understands two different input graph formats. The first format is suitable for sparse graphs and the second format is suitable for storing dense graphs (*i.e.*, graphs whose adjacency matrix contain mostly non-zeros). The format of these files are very similar to the corresponding formats for matrices, and the only difference is that they now store adjacency matrices which are square.

Note that CLUTO, automatically detects the format of the input file based on the first line of the file (*i.e.*, the sparse



(a)



(b)

Figure 14: Various visualizations generated by the *-plotcluster* parameter.

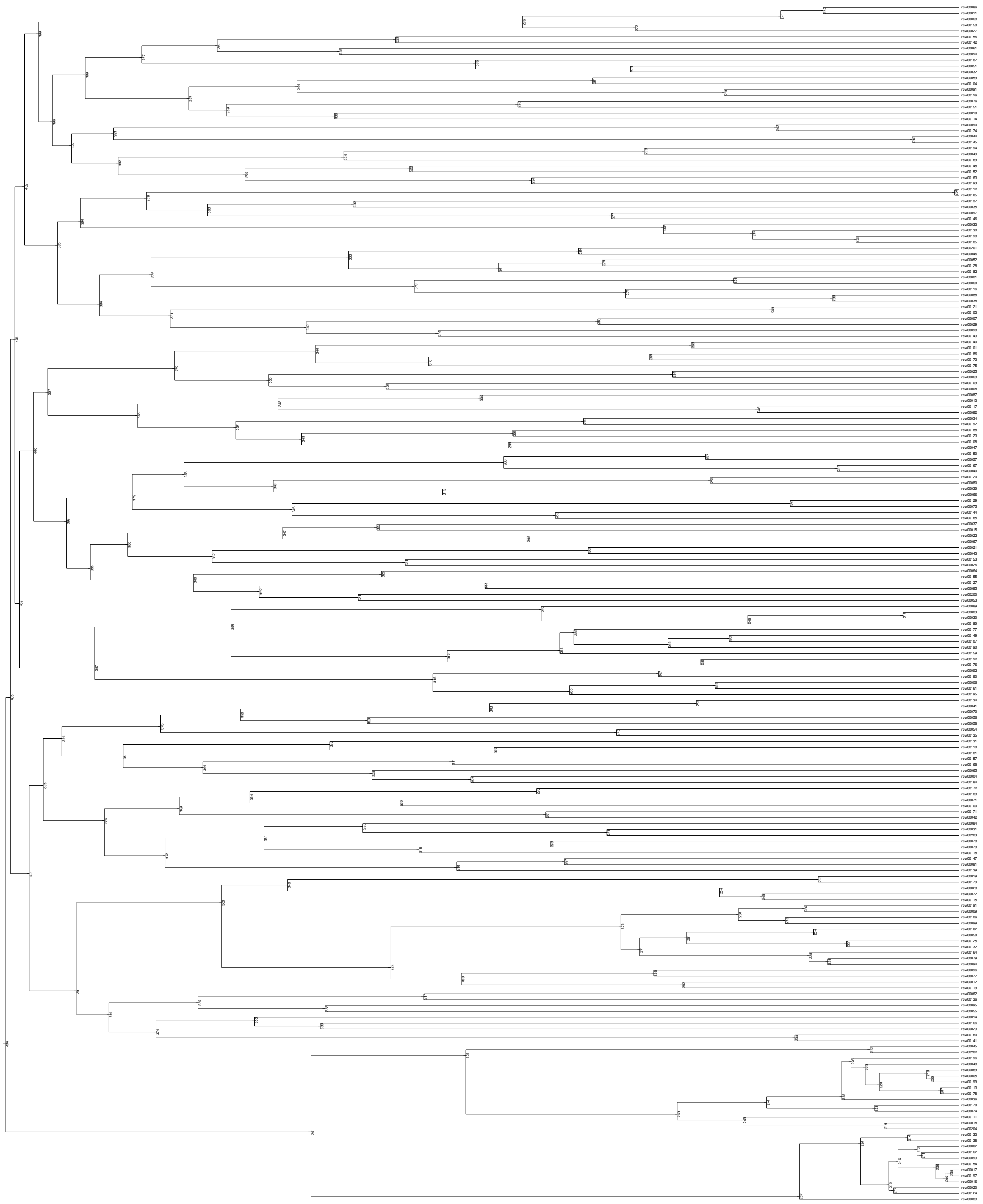


Figure 15: Various visualizations generated by the *-plottree* parameter.

	1.1			-0.5			0.2
1.4	0.4		-0.4				
		1.8			2.0		3.0
1.0							
	5.5		3.0			8.0	
		1.0		-1.0	2.0		
	3.5		-1.0	4.0		2.0	8.0

Matrix Input File

```
7  8  21
2  1.1  5  -0.5  8  0.2
1  1.4  2  0.4  4  -0.4
3  1.8  6  2.0  8  3.0
1  1.0
2  5.5  4  3.0  7  8.0
3  1.0  5  -1.0  6  2.0
2  3.5  4  -1.0  5  4.0  7  2.0  8  8.0
```

Figure 16: Storage format of a sample matrix.

graph format has two numbers whereas the dense graph format has one number).

Sparse Graph Format The adjacency matrix A of a sparse graph with n vertices is stored in a plain text file that contains $n+1$ lines. The first line contains information about the size of the graph, while the remaining n lines contain information for each row of A (*i.e.*, adjacency structure of the corresponding vertex). In CLUTO's sparse graph format only the non-zero entries of the adjacency matrix are stored.

The first line of the file contains exactly two numbers, all of which are integers. The first integer is the number of vertices in the graph (n) and the second integer is the number of edges in the graph (*i.e.*, the total number of non-zeros entries in A).

The remaining n lines store information about the actual non-zero structure of A. In particular, the $(i+1)$st line of the file contains information about the adjacency structure of the ith vertex (*i.e.*, the non-zero entries of the ith row of the adjacency matrix). The adjacency structure of each vertex is specified as a space-separated list of pairs. Each pair contains the number of the adjacent vertex followed by the similarity of the corresponding edge. The vertex numbers are assumed to be integers and their similarity values are assumed to be floating point numbers.

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

Dense Graph Format The adjacency matrix of a dense graph with n vertices is stored in a plain text file that contains $n+1$ lines. The first line stores information about the size of the graph, while the remaining n lines contain information for each row of the adjacency matrix. The first line of the file contains exactly one number, which is the number of vertices n of the graph. The remaining n lines store the values of the n columns of the adjacency matrix for each one of the vertices. In particular, each line contains exactly n space-separated floating point values, such that the ith value corresponds to the similarity to the ith vertex of the graph.

3.3.3 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.

3.3.4 Column Label File

As discussed in Section 3.1, when the *-clabelfile* parameter is used, the vcluster 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 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.

3.3.5 Row Class Label File

As discussed in Section 3.1, when the *-rclassfile* parameter is used, the vcluster 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.

3.4 Output File Formats

CLUTO's clustering programs can generate two different types of output files that store information about the clustering solution they have computed. 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 entire hierarchical agglomerative tree (when *-clmethod=agglo* or when the *-fulltree* option was specified(, or the agglomerative tree that was built on top of the computed clustering solution (when the *-showtree* option was specified). The format of these files is described in the following sections.

3.4.1 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/vertex belongs to. Cluster numbers run from zero to the number of clusters minus one.

In this case, CLUTO's clustering algorithms will not be able to assign all the objects to any of the clusters. In this case, the cluster number for that particular row/vertex will be set to -1. This usually happens for two reasons. First, CLUTO's vcluster program removes 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 which case they will not be clustered. Second, in the case of the graph-partitioning-based clustering algorithm, certain vertices of the graph may be pruned prior to clustering by using a combination of the *-edgeprune*, *-vtxprune*, or *-mincomponent* parameters.

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.

3.4.2 Tree File

The tree produced by performing a hierarchical agglomerative clustering on top of the k-way clustering solution produced by vcluster 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 10 will contain 19 lines, and each line will store the following numbers (one number per line): 10, 13, 16, 12, 10, 13, 15, 12, 11, 11, 14, 14, 17, 15, 16, 18, 17, 18, -1.

In addition to the parent of each node, CLUTO's tree file also outputs two numbers for each internal node the tree. The first number is 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. The second number is the change in the value of the criterion function achieved by combining the particular pair of clusters. Note that in the case of the traditional single-link, complete-link, and UPGMA agglomerative methods, the *gain* of the agglomeration is considered to be the weight of the *link* used in making the merging decisions.

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 Which Clustering Algorithm Should I Use?

If you have read CLUTO's manual up to this point you may start to wondering about which clustering algorithm to use for your application. Well, there is no correct answer, as it highly depends on the nature of your datasets and what constitutes meaningful clusters in your application. Nevertheless, this section attempts to clarify some of the "sweet spots" of CLUTO's various clustering algorithms and provide some general usage guidelines.

4.1 Cluster Types

We start our discussion by describing two different types of clusters that often arise in different application domains. What differentiates them is the relationship between the cluster's objects and the dimensions of their feature space. Note that this is by no means an exhaustive list of cluster types.

The first type of clusters contains objects that exhibit a strong pattern of conservation along a subset of their dimensions. That is, there is a subset of the original dimensions in which a large fraction of the objects agree. For example, if the dimensions correspond to words (or products), what that means is that a collection of documents (or customers) will form a cluster, if there exist a subset of terms (or products) that are present (or purchased) in a large fraction of the documents (or customers). You can actually see this type of clusters by looking at the visualization examples shown in Figure 11, as well as, the weights associated with the descriptive features that were output using the *-showfeatures* option in Figure 6. In the case of the visualizations, you can clearly see some of the dimensions (*i.e.*, columns) that are conserved in each cluster, and in the case of *-showfeatures* you can see that the top-5 terms in each cluster accounts for a large fraction of the similarity between the objects of each cluster.

This subset of dimensions is often referred to as a *subspace*, and the above stated property can be viewed as the cluster's objects and its associated dimensions forming a *dense subspace*. Of course, the number of dimensions in these dense subspaces, as well as, the density (*i.e.*, how large is the *fraction* of the objects that share the same dimensions) will be different from cluster to cluster. Exactly this variation in subspace size and density (and the fact that an object can be part of multiple disjoint or overlapping dense subspaces) is what complicates the problem of discovering this type of clusters. There are a number of application areas in which this type of clusters give rise to meaningful grouping of the objects (*i.e.*, domain experts will tend to agree that the clusters are correct). Such areas includes clustering documents based on the terms they contain, clustering customers based on the products they purchase, clustering genes based on their expression levels, clustering proteins based on the motifs they contain, *etc*.

The second type of clusters contains objects in which again there exist a subspace associated with that cluster. However, unlike the earlier case, in these clusters there will be sub-clusters that share a very small number of the subspace's dimension, but there will be *a strong path* within that cluster that will connect them. By "strong path" we mean that if A and B are two sub-clusters that share only a few dimensions, then there will be another set of sub-clusters $X_1, X_2, \ldots, X_k$, that belong to the cluster, such that each of the sub-cluster pairs (A, X_1), (X_1, X_2), ..., (X_k, B) will share many of the subspace's dimensions. What complicates cluster discovery in this setting is that the connections (*i.e.*, shared subspace dimensions) between sub-clusters within a particular cluster will tend to be of different strength. Examples of such clusters are the spatial clusters present in the two-dimensional datasets of Figure 3. In this case, the dimensions in our definition correspond to small ranges of the x and y-axis. With this in mind, we see that there are groups of points in the Π-shaped clusters that do not share either of the x or y ranges, However, there is a spatially contiguous set of points that connect them.

Our discussion so far focused on the relationship between the objects and their feature space. However, these two classes of clusters can also be understood in terms of the the object-to-object similarity graph. The first type of clusters will tend to contain objects in which the similarity between all pairs of objects will be high. On the other hand, in the second type of clusters there will be a lot of objects whose direct pairwise similarity will be quite low, but these objects will be connected by many paths that stay within the cluster that traverse high similarity edges. The names of these two cluster types were inspired by this similarity-based view, and they are referred to as *globular* and *transitive* clusters, respectively.

Matching Algorithms to Cluster Types CLUTO provides clustering algorithms for finding both of these types of clusters. In particular, the partitional clustering algorithms corresponding to “rb”, “rbr”, and “direct”, and the agglomerative algorithms “agglo” and “bagglo” that do not use the single-link criterion tend to find globular clusters. On the other hand, the agglomerative scheme with the single-link criterion and the graph-partitioning-based clustering algorithms tend to find transitive clusters. It should be noted that any of the algorithms can find either globular or transitive clusters provided that these clusters are sufficiently far away from each other.

The different clustering criterion functions used by the partitional and agglomerative clustering algorithms impact the extent to which the individual instance of the clustering algorithm is capable of finding globular clusters that contain clusters with different size consensus, or clusters whose average pair-wise similarity is different, as well as, the extent to which clusters can be of dramatically different sizes. The reader is referred to [6] for an analysis of these criterion functions.

4.2 Similarity Measures Between Objects

CLUTO’s clustering algorithms implemented by vcluster 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, the Pearson’s correlation coefficient, extended Jaccard coefficient, or a similarity derived from the Euclidean distance of these vectors. By using the cosine and correlation coefficient measures, then two objects are similar if their corresponding vectors[2] 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. On the other hand, the Euclidean distance does take into account both direction and magnitude. Finally, similarity based on extended Jaccard coefficient account both for angle, as well as, magnitude.

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 [6]. On the other hand, the Euclidean distance based similarity function is well-suited for finding clusters in the original feature space, as it is the case for the spatial clusters shown in Figure 3.

There are applications in which the provided similarity measures are not sufficient (*e.g.*, clustering sequence dataset). In such cases you have to use the scluster program in which you provide the pairwise similarities between the objects (you need to provide only the non-zero similarities). It is critical to ensure that the supplied similarities are reasonable, especially in the case of criterion driven partitional clustering (*i.e.*, for “rb”, “rbr”, and “direct”), as these approaches try to optimize the clustering criterion function, based only on these similarities. Some examples of bad similarity functions will be the ones in which there is a wide-range between the various similarity values, with some pairwise similarities being extremely large. In such cases, the optimal clustering solution (in terms of the criterion function) may just contain individual clusters for each such highly-similar pair of objects, with the rest of the objects assigned to one cluster.

4.3 Scalability of CLUTO’s Clustering Algorithms

The various clustering algorithms provided with CLUTO have different scalability characteristics. Table 2 summarizes the time- and space-complexity of some of the clustering algorithms.

Looking at these results we can see that in terms of time and memory, the most scalable method is vcluster’s repeated-bisecting algorithm that uses the cosine similarity function (*i.e.*, *-clmethod=rb*, *-sim=cos*). Our experiments showed that it can compute a 10-way partitioning of a dataset with 140K documents and 83K terms in less than five minutes on a Intel Xeon based workstation. The least scalable of the algorithms are the ones based on hierarchical agglomerative clustering. The critical aspect of these algorithms is that their memory requirements scale quadratic on the number of objects, and they cannot be used to cluster more than 5K-10K objects. However, if you do want to obtain a tree for a large dataset you should then use the *-fulltree* option that combines partitional and agglomerative clustering.

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

vcluster

Algorithm	Time Complexity	Space Complexity
-clmethod=rb, -sim=cos	$O(NNZ * \log(k))$	$O(NNZ)$
-clmethod=rb, -sim=corr	$O(n * m * \log(k))$	$O(n * m)$
-clmethod=direct, -sim=cos	$O(NNZ * k + m * k)$	$O(NNZ + m * k)$
-clmethod=direct, -sim=corr	$O(n * m * k)$	$O(n * m + m * k)$
-clmethod=agglo,	$O(n^2 * \log(n))$	$O(n^2)$
-clmethod=agglo, -crfun=[$\mathcal{I}_1$,$\mathcal{I}_2$]	$O(n^3)$	$O(n^2)$
-clmethod=graph,	$O(n^2 + n * NNbrs * \log(k))$	$O(nNNbrs)$

scluster

Algorithm	Time Complexity	Space Complexity
-clmethod=rb, -sim=cos	$O(NNZ * \log(k))$	$O(NNZ)$
-clmethod=rb, -sim=corr	$O(n * m * \log(k))$	$O(n * m)$
-clmethod=direct, -sim=cos	$O(NNZ * k + m * k)$	$O(NNZ + m * k)$
-clmethod=direct, -sim=corr	$O(n * m * k)$	$O(n * m + m * k)$
-clmethod=agglo,	$O(n^2 * \log(n))$	$O(n^2)$
-clmethod=agglo, -crfun=[$\mathcal{I}_1$,$\mathcal{I}_2$]	$O(n^3)$	$O(n^2)$
-clmethod=graph,	$O(n * NNbrs * \log(k))$	$O(nNNbrs)$

Table 2: The complexity of CLUTO's clustering algorithms. The meaning of the various quantities are as follows: n is the number of objects to be clustered, m is the number of dimensions, *NNZ* is the number of non-zeros in the input matrix or similarity matrix, *NNbrs* is the number of neighbors in the nearest-neighbor graph.

5 CLUTO's Library Interface

The functionality provided by CLUTO's vcluster and scluster programs 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.

5.1 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.

5.2 Matrix and Graph Data Structure

Most of the routines in CLUTO's library take, as input, the objects to be clustered in the form of a matrix. For some routines this matrix corresponds to the feature-space representation of the objects, that is, the rows are the objects and the columns are the features (just like the matrix-file for the vcluster program). Whereas for some other routines, this matrix corresponds to the adjacency matrix of the similarity graph between the objects, that is, both the rows and the columns of the matrix correspond to the vertices in the graph (just like the graph-file for the scluster program).

Even though these two type of matrices represent entirely different information, they are provided to CLUTO's routines using the same data structure. This is primarily because the adjacency matrix of a graph is, after all, a matrix which just happens to have the same number of rows and columns.

CLUTO's routines support both sparse and dense matrices using the same set of data structures.

Sparse Matrix and Graph 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]`.

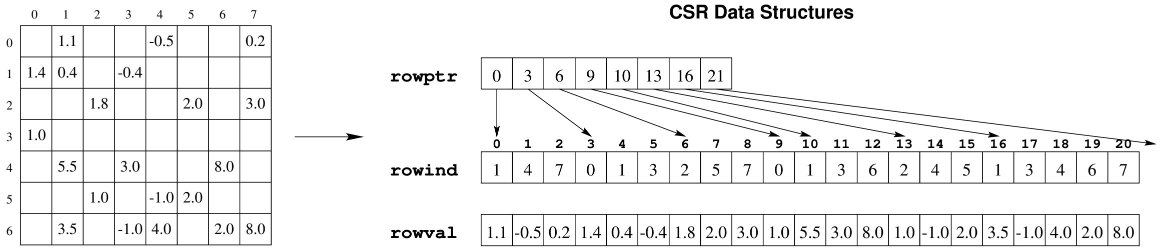


Figure 17: An example of the CSR format for storing sparse matrices.

Figure 17 illustrates the CSR format for the sparse matrix used earlier to illustrated the format of the matrix file used by vcluster. 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]`.

5.3 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.

5.3.1 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 vcluster. 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 vcluster and scluster.

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

CLUTO_SIM_EDISTANCE The similarity between the objects is computed to be inversely related to their Euclidean distance. In particular, if $d_{i,j}$ is the distance between two objects, and $d_{\max}$ is the maximum distance between any two objects in the dataset, the similarity between these objects is set to be

$$\mathrm{sim}(i, j) = 1 - \frac{d_{i,j}}{1.0 + d_{\max}}.$$

CLUTO_SIM_EJACCARD The similarity between the objects is computed using the extended Jaccard coefficient of their vectors. If u and v are the vectors of two objects, their extended Jaccard coefficient is:

$$\mathrm{sim}_{ejacc}(u, v) = \frac{u^t v}{\|u\| + \|v\| - u^t v}.$$

5.3.2 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 vcluster and scluster. The possible values for the *crfun* parameter are the following:

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

CLUTO_CLFUN_I2 Selects the I2 ($\mathcal{I}_2$) criterion function.

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

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

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

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

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

CLUTO_CLFUN_SLINK Selects the traditional single-link merging criterion.

CLUTO_CLFUN_SLINK_W Selects the weighted single-link merging criterion, in which the initial similarity between two clusters is scaled by the sum of the similarities between the objects of the cluster.

CLUTO_CLFUN_CLINK Selects the traditional complete-link merging criterion.

CLUTO_CLFUN_CLINK_W Selects the weighted complete-link merging criterion, in which the initial similarity between two clusters is scaled by the sum of the similarities between the objects of the cluster.

CLUTO_CLFUN_UPGMA Selects the traditional UPGMA merging criterion.

5.3.3 The *cstype* Parameter

This parameter specifies the method to be used for selecting the next cluster to be bisected by CLUTO's repeated-bisecting- and graph-partitioning-based clustering algorithms. This parameter is similar to the *-cstype* option of vcluster and scluster. The possible values for the *cstype* parameter are the following:

CLUTO_CSTYPE_LARGEFIRST Selects to bisect the largest cluster from the current clustering solution.

CLUTO_CSTYPE_BESTFIRST	Selects to bisect the cluster that will lead to the best value of the clustering criterion function that is guides the clustering process.
CLUTO_CSTYPE_LARGESUBSPACEFIRST	Selects to bisect the cluster that will lead to the largest reduction on the number of the subspace dimensions of this cluster.

5.4 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*.

5.4.1 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 vcluster. 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.
CLUTO_ROWMODEL_MAXTF	The columns of each row are scaled so their values are between 0.5 and 1.0.
CLUTO_ROWMODEL_SQRT	The columns of each row are scaled to be equal to the square root of their actual values.
CLUTO_ROWMODEL_LOG	The columns of each row are scaled to be equal to the log of their actual values.

5.4.2 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 vcluster. 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 vcluster's section.

5.4.3 The *grmodel* Parameter

This parameter specifies the type of k-nearest neighbor graph that will be built by CLUTO's graph-partitioning based clustering algorithms. This parameter is similar to the *-grmodel* option of vcluster and scluster. The possible values for this parameter are:

CLUTO_GRMODEL_SYMETRIC_DIRECT	An edge between two vertices u and v is included if and only if they are in the nearest-neighbor list of each other. The weight of this edge is set equal to the similarity of the objects.
CLUTO_GRMODEL_ASSYMETRIC_DIRECT	An edge between two vertices u and v is included as long as one of them is in the nearest-neighbor list of the other. The weight of this edge is set equal to the similarity of the objects.

CLUTO_GRMODEL_SYMETRIC_LINK	An edge between two vertices u and v is included if and only if they are in the nearest-neighbor list of each other. The weight of this edge was set equal to the number of neighbors that vertices u and v have in common.
CLUTO_GRMODEL_ASSYMETRIC_LINK	An edge between two vertices u and v is included as long as one of them is in the nearest-neighbor list of the other. The weight of this edge was set equal to the number of neighbors that vertices u and v have in common.
CLUTO_GRMODEL_NONE	The supplied graph is used as is.

5.4.4 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 vcluster.

5.4.5 The *edgeprune* Parameter

This parameter controls how the edges in the graph-partitioning clustering algorithms will be pruned based on the link-connectivity of their incident vertices. Please refer to the discussion of CLUTO's *-edgeprune* for further details. A value of -1 suppresses edge-pruning.

5.4.6 The *vtxprune* Parameter

This parameter controls how outlier vertices in the graph-partitioning clustering algorithms will be pruned based on their degree. Please refer to the discussion of CLUTO's *-vtxprune* for further details. A value of -1 suppresses vertex-pruning.

5.5 Debugging Parameter

Most of CLUTO's routines take as input a parameter called *dbglvl* that controls the amount of information to be printed. This is used for internal purposes and should be set to 0, which suppresses any debugging output.

5.6 Clustering Routines

5.6.1 CLUTO_VP_ClusterDirect

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 *-clmethod=direct* clustering method of the vcluster 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 5.2.

simfun, crfun
The clustering parameters whose meaning and possible values are described in Section 5.3.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

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 vcluster'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 5.5.

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

5.6.2 CLUTO_VP_ClusterRB

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 cstype, 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 *-clmethod=rb* and *-clmethod=rbr* clustering methods of the vcluster 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 5.2.

simfun, crfun, cstype
The clustering parameters whose meaning and possible values are described in Section 5.3.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

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 5.5.

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

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).

5.6.3 CLUTO_VP_GraphClusterRB

int **CLUTO_VP_GraphClusterRB** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval,
int simfun, int rowmodel, int colmodel, float colprune, int grmodel,
int nnbrs, float edgeprune, float vtxprune, int mincmp,
int ntrials, int seed, int cstype, int dbglvl, int nclusters,
int *part, float *crvalue)

Description

Used to cluster a matrix into a specified (k) number of clusters using a graph-partitioning-based clustering algorithm that computes the k-way by performing a sequence of repeated min-cut bisections. Provides the functionality of the *-clmethod=graph* clustering method of the vcluster 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 5.2.

simfun, crfun, cstype
: The clustering parameters whose meaning and possible values are described in Section 5.3.

rowmodel, colmodel, colprune, vtxprune, edgeprune
: The object modeling parameters whose meaning and possible values are described in Section 5.4.

nnbrs The number of neighbors of each object that will be used to create the nearest neighbor graph.

mincmp The size of the minimum connect component that will be pruned prior to clustering.

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.

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

dbglvl The debugging parameter whose meaning and possible values are described in Section 5.5.

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.

crvalue This is a variable that upon returns stores the edge-cut of the clustering solution.

Returned Value

Returns the number of clusters that it found. This number will be equal to the number of desired clusters plus the number of connected components in the graph.

Note

5.6.4 CLUTO_VA_Cluster

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, float *gains)

Description

Used to cluster a matrix into a specified (k) number of clusters using a hierarchical agglomerative clustering algorithm. Provides the functionality of the *-clmethod=agglo* clustering method of the vcluster 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 5.2.

simfun, crfun
: The clustering parameters whose meaning and possible values are described in Section 5.3.

rowmodel, colmodel, colprune
: The object modeling parameters whose meaning and possible values are described in Section 5.4.

dbglvl The debugging parameter whose meaning and possible values are described in Section 5.5.

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.

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 ith 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 ith 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.

gains This is an array of size *2*nrows* that upon successful completion stores the gains in the value of the criterion function resulted by the merging pairs of clusters. In particular, *gains[i]* stores the gain achieved by merging the clusters that are the children of the ith 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

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.

5.6.5 CLUTO_VA_ClusterBiased

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

Description

Used to cluster a matrix into a specified (k) number of clusters using a hierarchical agglomerative clustering algorithm that is biased by a partitionally computed clustering solution. Provides the functionality of the -*clmethod=bagglo* clustering method of the vcluster 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 5.2.

simfun, crfun
The clustering parameters whose meaning and possible values are described in Section 5.3.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

dbglvl The debugging parameter whose meaning and possible values are described in Section 5.5.

npclusters
The number of clusters for which the partitional clustering solution will be computed. In the case of the *-clmethod=bagglo* this is set automatically to $\sqrt{\text{nrows}}$.

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.

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 ith 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 ith 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.

gains This is an array of size *2*nrows* that upon successful completion stores the gains in the value of the criterion function resulted by the merging pairs of clusters. In particular, *gains[i]* stores the gain achieved by merging the clusters that are the children of the ith 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

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

5.6.6 CLUTO_SP_ClusterDirect

void **CLUTO_SP_ClusterDirect** (int nrows, int *rowptr, int *rowind, float *rowval, int crfun,
int ntrials, int niter, int seed, int dbglvl, int nclusters, int *part)

Description

Used to cluster a graph into a specified (k) number of clusters using a partitional clustering algorithm that computes the k-way clustering directly. Provides the functionality of the *-clmethod=direct* clustering method of the scluster program.

Input Parameters

nrows The number of rows of the input adjacency matrix whose rows store the adjacency structure of the between object similarity graph.

rowptr, rowind, rowval
The matrix itself in the format described in Section 5.2.

crfun The clustering criterion function whose meaning and possible values are described in Section 5.3.

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 vcluster'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 5.5.

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

5.6.7 CLUTO_SP_ClusterRB

void **CLUTO_SP_ClusterRB** (int nrows, int *rowptr, int *rowind, float *rowval, int crfun
int ntrials, int niter, int seed, int cstype, 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 *-clmethod=rb* and *-clmethod=rbr* clustering methods of the scluster program.

Input Parameters

nrows The number of rows of the input adjacency matrix whose rows store the adjacency structure of the between-object similarity graph.

rowptr, rowind, rowval
The matrix itself in the format described in Section 5.2.

crfun, cstype
The clustering parameters whose meaning and possible values are described in Section 5.3.

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 5.5.

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

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

5.6.8 CLUTO_SP_GraphClusterRB

int **CLUTO_SP_GraphClusterRB** (int nrows, int *rowptr, int *rowind, float *rowval, int nnbrs, float edgeprune, float vtxprune, int mincmp, int ntrials, int seed, int cstype, int dbglvl, int nclusters, int *part, float *crvalue)

Description

Used to cluster a matrix into a specified (k) number of clusters using a graph-partitioning-based clustering algorithm that computes the k-way by performing a sequence of repeated min-cut bisections. Provides the functionality of the *-clmethod=graph* clustering method of the scluster program.

Input Parameters

nrows The number of rows of the input adjacency matrix whose rows store the adjacency structure of the between-object similarity graph.

rowptr, rowind, rowval
The matrix itself in the format described in Section 5.2.

cstype The clustering parameters whose meaning and possible values are described in Section 5.3.

vtxprune, edgeprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

nnbrs The number of neighbors used in the edge- and vertex-pruning calculations. Note that in this routine, this variable does not control the number of neighbors in the graph.

mincmp The size of the minimum connect component that will be pruned prior to clustering.

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.

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

dbglvl The debugging parameter whose meaning and possible values are described in Section 5.5.

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.

crvalue This is a variable that upon returns stores the edge-cut of the clustering solution.

Returned Value

Returns the number of clusters that it found. This number will be equal to the number of desired clusters plus the number of connected components in the graph.

Note

5.6.9 CLUTO_SA_Cluster

void **CLUTO_SA_Cluster** (int nrows, int *rowptr, int *rowind, float *rowval, int crfun,
int dbglvl, int nclusters, int *part, int *ptree, float *tsims, float *gains)

Description

Used to cluster a matrix into a specified (k) number of clusters using a hierarchical agglomerative clustering algorithm. Provides the functionality of the *-clmethod=agglo* clustering method of the scluster program.

Input Parameters

nrows The number of rows of the input adjacency matrix whose rows store the adjacency structure of the between-object similarity graph.

rowptr, rowind, rowval
The matrix itself in the format described in Section 5.2.

crfun The clustering parameters whose meaning and possible values are described in Section 5.3.

dbglvl The debugging parameter whose meaning and possible values are described in Section 5.5.

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.

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 ith 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 ith 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.

gains This is an array of size *2*nrows* that upon successful completion stores the gains in the value of the criterion function resulted by the merging pairs of clusters. In particular, *gains[i]* stores the gain achieved by merging the clusters that are the children of the ith 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

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

5.6.10 CLUTO_V_BuildTree

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 dbglvl, int nclusters, int *part, int *ptree, float *tsims, float *gains)

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 vcluster 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 vcluster 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 5.2.

simfun, crfun
The clustering parameters whose meaning and possible values are described in Section 5.3.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

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.

dbglvl The debugging parameter whose meaning and possible values are described in Section 5.5.

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.

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 ith 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 ith 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.

gains 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 gains in the value of the criterion function resulted by the merging pairs of clusters. In particular, *gains[i]* stores the gain achieved by merging the clusters that are the children of the ith 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

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.

5.6.11 CLUTO_S_BuildTree

void **CLUTO_S_BuildTree** (int nrows, int *rowptr, int *rowind, float *rowval, int crfun, int treetype, int dbglvl, int nclusters, int *part, int *ptree, float *tsims, float *gains)

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 scluster 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 scluster is specified. The hierarchical agglomerative tree is build so that it optimizes a particular clustering criterion function.

Input Parameters

nrows The number of rows of the input adjacency matrix whose rows store the adjacency structure of the between-object similarity graph.

rowptr, rowind, rowval
The matrix itself in the format described in Section 5.2.

crfun The clustering parameters whose meaning and possible values are described in Section 5.3.

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.

dbglvl The debugging parameter whose meaning and possible values are described in Section 5.5.

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.

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 ith 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 ith 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.

gains 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 gains in the value of the criterion function resulted by the merging pairs of clusters. In particular, *gains[i]* stores the gain achieved by merging the clusters that are the children of the ith 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

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.

5.7 Graph Creation Routines

5.7.1 CLUTO_V_GetGraph

int **CLUTO_V_GetGraph** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval,
int simfun, int rowmodel, int colmodel, float colprune, int grmodel,
int nnbrs, int dbglvl, int **growptr, int **growind, float **growval)

Description

Used to create a nearest-neighbor graph of the set of objects. This is graph can be used as input to the graph-partitioning based clustering algorithm (CLUTO_SP_GraphClusterRB).

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 5.2.

simfun The method used to compute the similarity between objects, whose meaning and possible values are described in Section 5.3.1.

rowmodel, colmodel, grmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

nnbrs The number of neighbors of each object that will be used to create the nearest neighbor graph.

dbglvl The debugging parameter whose meaning and possible values are described in Section 5.5.

Output Parameters

growptr, growind, growval
These are three arrays storing the computed graph in the CSR matrix format. Memory for these arrays are allocated within CLUTO's library. However, the application is responsible for freeing this memory.

Note

5.7.2 CLUTO_S_GetGraph

int **CLUTO_S_GetGraph** (int nrows, int *rowptr, int *rowind, float *rowval, int grmodel,
int nnbrs, int dbglvl, int **growptr, int **growind, float **growval)

Description

Used to create a nearest-neighbor graph of the set of objects. This is graph can be used as input to the graph-partitioning based clustering algorithm (CLUTO_SP_GraphClusterRB).

Input Parameters

nrows The number of rows of the adjacency matrix (*i.e.*, the number of vertices in the graph).

rowptr, rowind, rowval
The matrix itself in the format described in Section 5.2.

grmodel The type of graph to be constructed. The meaning and possible values are described in Section 5.4.

nnbrs The number of neighbors of each object that will be used to create the nearest neighbor graph.

dbglvl The debugging parameter whose meaning and possible values are described in Section 5.5.

Output Parameters

growptr, growind, growval
These are three arrays storing the computed *nrows*-vertex graph in the CSR matrix format. Memory for these arrays are allocated within CLUTO's library. However, the application is responsible for freeing this memory.

Note

5.8 Cluster Statistics Routines

5.8.1 CLUTO_V_GetSolutionQuality

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 5.2.

simfun, crfun
The clustering parameters whose meaning and possible values are described in Section 5.3.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

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.

Returned Value

This function returns the value of the clustering criterion function of the supplied clustering solution. Please refer to [6] for the exact definitions of these criterion functions.

Note

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.

5.8.2 CLUTO_S_GetSolutionQuality

float **CLUTO_S_GetSolutionQuality** (int nrows, int *rowptr, int *rowind, float *rowval, int crfun,
int nclusters, int *part)

Description

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

Input Parameters

nrows 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 5.2.

crfun The clustering parameters whose meaning and possible values are described in Section 5.3.

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.

Returned Value

This function returns the value of the clustering criterion function of the supplied clustering solution. Please refer to [6] for the exact definitions of these criterion functions.

Note

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.

5.8.3 CLUTO_V_GetClusterStats

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 5.2.

simfun The clustering similarity function whose meaning and possible values are described in Section 5.3.1.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

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.

Output Parameters

pwgts An array of size *nclusters* that returns the sizes of the different clusters. In particular, the size of the ith 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 ith 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 ith 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 ith object is returned in izscores[i]. The internal z-score of each object is described in the discussion of the *-zscores* option of vcluster. 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 ith 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 ith 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 ith object is returned in ezscores[i]. The external z-score of each object is described in the discussion of the *-zscores* option of vcluster. 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.

5.8.4 CLUTO_S_GetClusterStats

void **CLUTO_S_GetClusterStats** (int nrows, int *rowptr, int *rowind, float *rowval, 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 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 5.2.

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.

Output Parameters

pwgts An array of size *nclusters* that returns the sizes of the different clusters. In particular, the size of the ith 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 ith 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 ith 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 ith object is returned in izscores[i]. The internal z-score of each object is described in the discussion of the *-zscores* option of vcluster. 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 ith 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 ith 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 ith object is returned in ezscores[i]. The external z-score of each object is described in the discussion of the *-zscores* option of vcluster. The application is responsible for allocating the memory for this array.

Note

5.8.5 CLUTO_V_GetClusterFeatures

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 vcluster 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 5.2.

simfun The clustering similarity function whose meaning and possible values are described in Section 5.3.1.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

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.

5.8.6 CLUTO_V_GetClusterSummaries

void **CLUTO_V_GetClusterSummaries** (int nrows, int ncols, int *rowptr, int *rowind, float *rowval, int simfun, int rowmodel, int colmodel, float colprune, int nclusters, int *part, int sumtype, int nfeatures, int *r_nsum, int **r_spid, float **r_swgt, int **r_sumptr, int **r_sumind)

Description

Returns sets of features that frequently co-occur within the objects of each cluster. It provides the functionality of the *-showsummaries* option of the vcluster 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 5.2.

simfun The clustering similarity function whose meaning and possible values are described in Section 5.3.1.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

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.

sumtype
Specifies the type of summaries that needs to be computed. The possible values for this parameter are:

CLUTO_SUMTYPE_MAXCLIQUES	Returns the features that form maximal cliques in the feature-to-feature co-occurrence graph.
CLUTO_SUMTYPE_MAXITEMSETS	Returns the features that occur frequently in the objects of each cluster. A frequent itemset is returned if it is maximal or if its frequency is much higher than the frequency of its maximal itemsets.

nfeatures The number of the most descriptive features for which the summarization will be performed.

Output Parameters

r_nsum This is the number of discovered summaries.

r_spid, r_swgt, r_sumptr, r_sumind
This is a set of four arrays that store information about the discovered summaries. Since the number of summaries is dataset and clustering-solution dependent, CLUTO allocates memory for these arrays internally, and returns them to the application. This is why all of these arrays are "**". The application is responsible for deallocating the memory for these arrays.

The *r_spid* and *r_swgt* arrays are of size *r_nsum* and each entry of these arrays is associated with the ith summary. In particular, *r_spid[i]* stores the cluster number that the ith summary belongs too, and *r_swgt[i]* stores a weight associated with that summary. If the summaries were computed using the *maxcliques* option, this weight represents the density of the features in the clique. If the summaries were computed using the *maxitemsets* option, this weight represents the support of the corresponding itemset.

The arrays *r_sumptr* and *r_sumind* store the actual features of the various summaries. The *r_sumptr* array is of size *r_nsum+1* and the features of the ith summary is stored in *r_sumind* starting at location *r_sumptr[i]* up to (but not including) location *r_sumptr[i+1]*.

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.

This routine will produce meaningful results only for sparse and high-dimensional datasets.

5.8.7 CLUTO_V_GetTreeStats

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 5.2.

simfun The clustering similarity function whose meaning and possible values are described in Section 5.3.1.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

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.

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 ith 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 ith 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 ith 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.

5.8.8 CLUTO_V_GetTreeFeatures

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 vcluster 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 5.2.

simfun The clustering similarity function whose meaning and possible values are described in Section 5.3.1.

rowmodel, colmodel, colprune
The object modeling parameters whose meaning and possible values are described in Section 5.4.

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 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*, *nclusters*, *part*, and *ptree* parameters should be identical to those used to compute the clustering solution and build the hierarchical agglomerative tree.

6 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 CLUTO will be made available on WWW at *http://www.cs.umn.edu/˜karypis/cluto*.

7 Copyright Notice and Usage Terms

References

[1] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. In *Proc. of 1998 ACM-SIGMOD Int. Conf. on Management of Data*, 1998.

[2] G. Karypis, E.H. Han, and V. Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*, 32(8):68–75, 1999.

[3] G. Karypis and V. Kumar. hMETIS 1.5: A hypergraph partitioning package. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL *http://www.cs.umn.edu/˜metis*.

[4] G. Karypis and V. Kumar. METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL *http://www.cs.umn.edu/˜metis*.

[5] Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *CIKM*, 2002.

[6] 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*.