CompClustTk
Manual & Tutorial

Brandon King
Diane Trout

Version 1.0
May 27, 2005

Contents

1 Introduction

1.1 Purpose

CompClust provides utilities designed around the needs of biologists explore various types of array data, such as microarray or ChIP array.

CompClust focuses on gaining a more quantitative and qualitative understanding of clustering results and the relationships between them. The software was initially developed as a set of python modules designed to provide a convenient environment for interactive use from the python interpreter and for then automating analysis as python scripts. The generality of these modules has proved to be useful in many applications within our lab for bioinformatics and genomic analysis. Although the most powerful and flexible way to use CompClust is directly via the python command line interface we have constructed CompClustTK and CompClustWeb as more familiar graphical user interface (GUI) to many of the most useful analysis tools. The GUIs focus on the analysis tools for comparative cluster analysis as described in (**?**).

This tutorial contains general introductory information for CompClust as well as specific information on how to use CompClustTk. Other tutorials and documentation have been written as an introduction to using CompClust via the python interpreter directly.

1.2 CompClust Background

CompClust is a Python package written using the pyMLX and IPlot APIs. It provides software tools to explore and quantify relationships between clustering results. Its development has been largely built around requirements for microarray data analysis, but can be easily used for other types of biological array data and that of other scientific domains.

Briefly, pyMLX provides efficient and convenient execution of many clustering algorithms using an extendable library of algorithms. It also provides many-to-many linkages between data features and annotations (such as cluster labels, gene names, gene ontology information, etc.) These linkages persistent through user data manipulation. IPlot provides an abstraction of the plotting process in which any arbitrary feature or derived feature of the data can be projected onto any feature of the plot, including the X,Y coordinates of points, marker symbol, marker size, marker/line color, etc. These plots are intrinsically linked to the dataset, the View and the Labeling classes found within pyMLX.

1.3 Where to get help?

If you need help with CompClustTk or CompClust visit http://woldlab.caltech.edu/compClust/.

Or you may send e-mail to Chris Hart (hartATcaltech.edu) or Brandon King (kingbATcaltech.edu).

1.4 What's New

To make the plots between the trajectory summary and the confusion matrix more consistant, the "Plot All" button on the Trajectory Summary plot was removed and now left-clicking on either the trajectory summary plot or confusion matrix cells will bring up the appropriate detail plot.

2 Tutorial Background

2.1 Microarray Data

Users unfamiliar with microarrays in general are encouraged to familiarize themselves with the the biology and technology behind the data. For this tutorial we assume the microarray data is gene expression measurements across several different conditions (eg. Different time points, different cell types, different treatments, etc.). The data will be needed to formatted into a gene expression data matrix, where each row is represents a gene and each column represents a different condition. A row in that matrix represents a gene expression vector, or the expression measurements for that gene across every assayed condition.

2.2 CompClust Datasets

CompClust Datasets contain vectors of data, which can be loaded into CompClustTk through a simple tab delimited text format.

Take microarray data for example. If you've done an experiment with four time points; hours 1, 2, 4, and 8. The columns of your data set are the time points and the rows are the individual genes (see below).

# Hour 1	Hour 2	Hour 4	Hour 8
0.72	0.56	0.32	0.06
0.01	0.15	0.80	0.73
0.97	0.95	0.91	0.94

Loading the above data set into CompClustTk and then running a Trajectory Summary plot (see section 3.5.2), you would see that the first gene's trajectory (vector) starts high and gets lower of the four time points. Formatting your data into a tab delimited formated similar to the one shown above will allow you to load your own data sets into CompClustTk.

2.3 CompClust Labelings

Adding a new labeling to any data set is fairly easy. All you need to do is make a tab delimited text file with either one row or one column depending on what type of labeling is appropriate. The only restriction is that labeling must be the same dimensions as it's data set.

For example, if you wanted to add a 'Gene Name' labeling to the data set in section 2.2. You would need a row labeling... i.e. one column with three labels to match the three rows in the data set. Below is an example of this labeling.

Gene Name 1
Gene Name 2
Gene Name 3

If you wanted to make your own cluster labeling (group labeling), you would reuse the same label in one or more rows. For example if I wanted to create a cluster labeling which groups Gene 1 and Gene 2 in one group and Gene 3 in another group, I would create the following row labeling.

Cluster 1
Cluster 1
Cluster 2

One may wish to keep around the time point hours as column labeling as well. To do this, create a tab delimited text file with one row as show in the column label below.

Hour 1	Hour 2
Hour 4	Hour 8

In actuality, labeling files can be in either in row form, as one label per row, or in column form as one label per tab separated column.

One of the beauties of CompClust is you can attach as many labels as you can think of. In CompClustTk you will see dialogs asking you to select cluster labelings, which are row labelings which separates your data into groups. And you will be requested for primary and secondary labelings, which are basically arbitrary row labelings which you may wish to attach. For example, when viewing gene expression data in a plot, you may wish to attach a primary labeling of gene names and a secondary labeling of descriptions.

Column labelings currently can only be taking advantage of by using CompClust from Python, but in the future, these features may be exposed in CompClustTk.

2.4 Cho Example Data

CompClustTk uses example data collected from Cho et. al., 1998. Briefly they synchronized yeast cells using a CDC28 temperature sensitive mutant. After releasing the yeast cells from arrest they collected RNA from the cells every 10 minutes as the cells underwent two rounds of cell division. Using Affymetrix arrays they assayed the gene expression profile of every gene in yeast during this experiment (Cho et al., 1998). The resulting gene expression matrix has roughly 6000 genes by 17 time points. We provide a subset of this matrix which includes a total of 380 genes that were both selected by the authors to exhibit cell cycle dependency and meet a minimal noise threshold (Hart et al., 2005). Hart et. al. 2005 provides a introduction and theoretical basis for these tools and also provides a case study highlighted the types of biological insights that can be gleaned from analysis similar to those described here.

3 CompClustTk

3.1 Introduction

CompClustTk was designed to expose the basic functionality of CompClust. The idea is to bring the CompClust analysis environment to the biologist. Previously, a basic knowledge of Python programming was required in order to use CompClust. This is still true for some of the most advanced analysis one may wish to do.

We hope that CompClustTk will simplify learning to use CompClust by allowing the user to use CompClust without knowing any Python. If you find CompClustTk too limiting, there are a few tools which will help you to adjust to using Python along with the GUI to do more advanced analysis—whenever you trigger an action, the Python code you would have used to do the same thing in pure Python will be stored in the 'Analysis History' section (View—Toggle Analysis History).

For those of you who are feeling daring or just don't like GUIs that much, you can use iPython to access the internals of the GUI, including any data sets or labelings which you have loaded.

In the following sections of this tutorial, we will be using the Cho et. al., 1998 Cell Cycling data mentioned above located in your CompClustTk/Examples/ChoCellCycling directory.

3.2 Load Data Set

The first thing we need to do after loading CompClustTk is to load a data set to work with. Click on 'File—Load Data' Set from the menu.

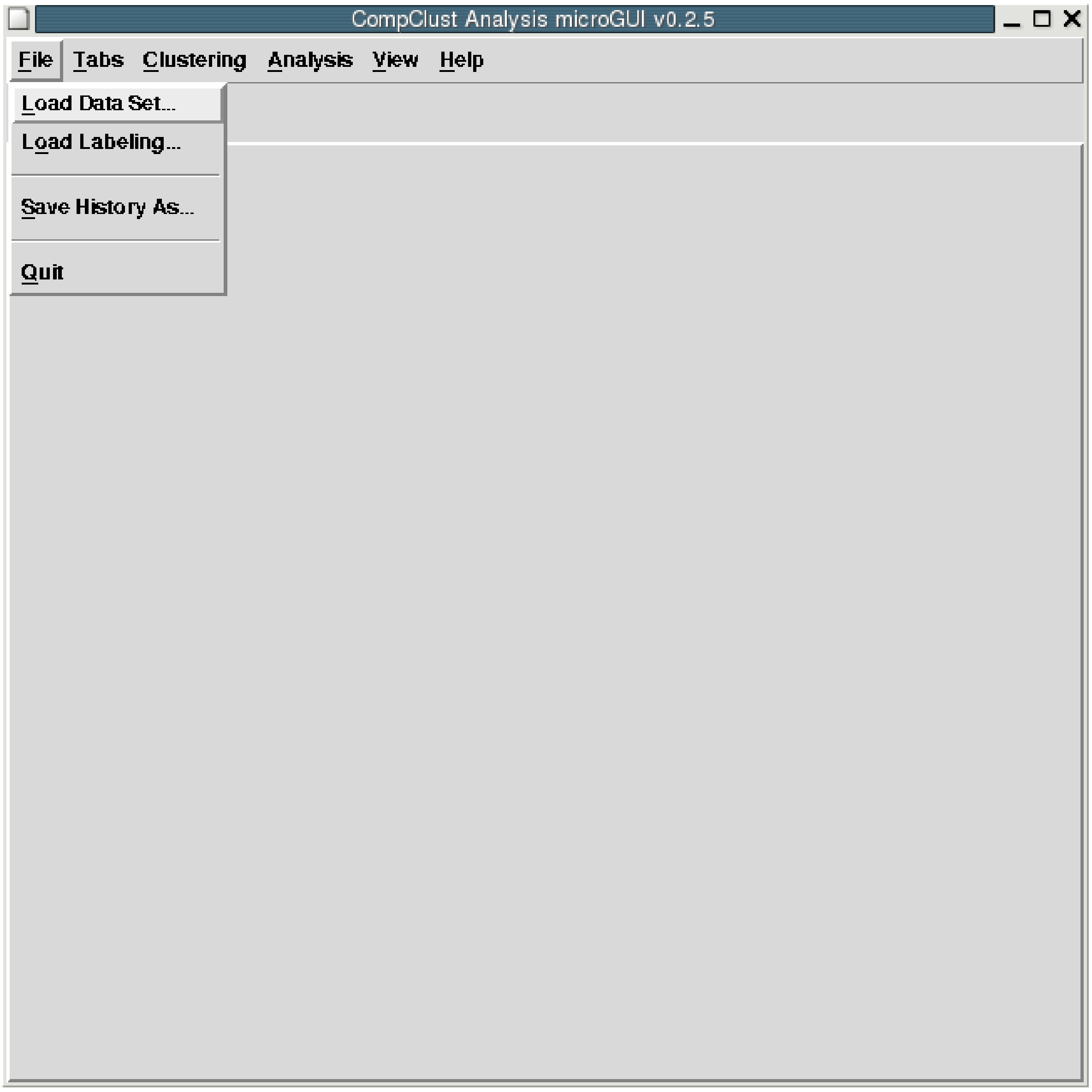
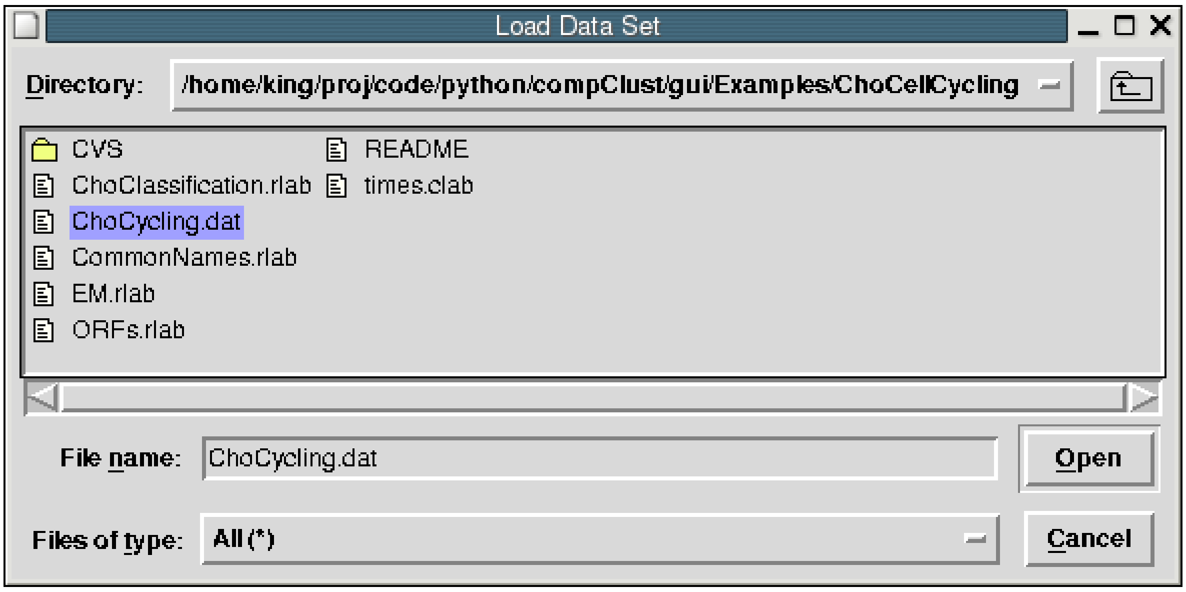


Figure 1: CompClustTk File — Load Data Set

Locate the file named 'ChoCycling.dat' and click 'Open'.



/home/king/proj/code/python/compClust/gui/Examples/ChoCellCycling

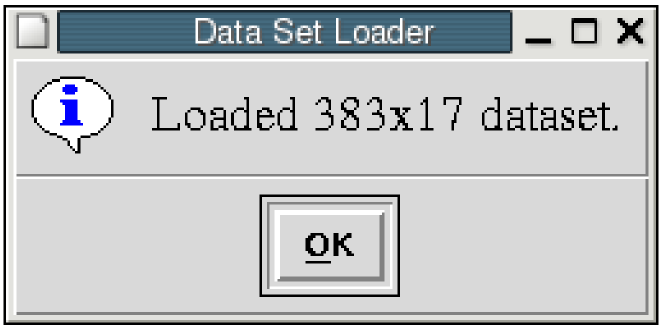
CVS
ChoClassification.rlab
ChoCycling.dat
CommonNames.rlab
EM.rlab
ORFs.rlab
README
times.clab

File name: ChoCycling.dat

Files of type: All (*)

Figure 2: Load Data Set Dialog

If everything went well, you should see a dialog box like the one below telling you the dimensions of the data set you have just loaded. Make sure that the numbers look reasonable, otherwise it may be a sign that your data set was not formatted properly.



Loaded 383x17 dataset.

Figure 3: Load Data Set Complete

3.3 Attach Labelings

Once you have successfully loaded your data set, it will be useful to attach additional information to the data set. The generic name for these annotations is ’Labelings’.

First, let’s load the Gene row Labeling so we will be able to know which rows (vectors) represent which genes. Select ’File—Load Labeling’ as shown below.

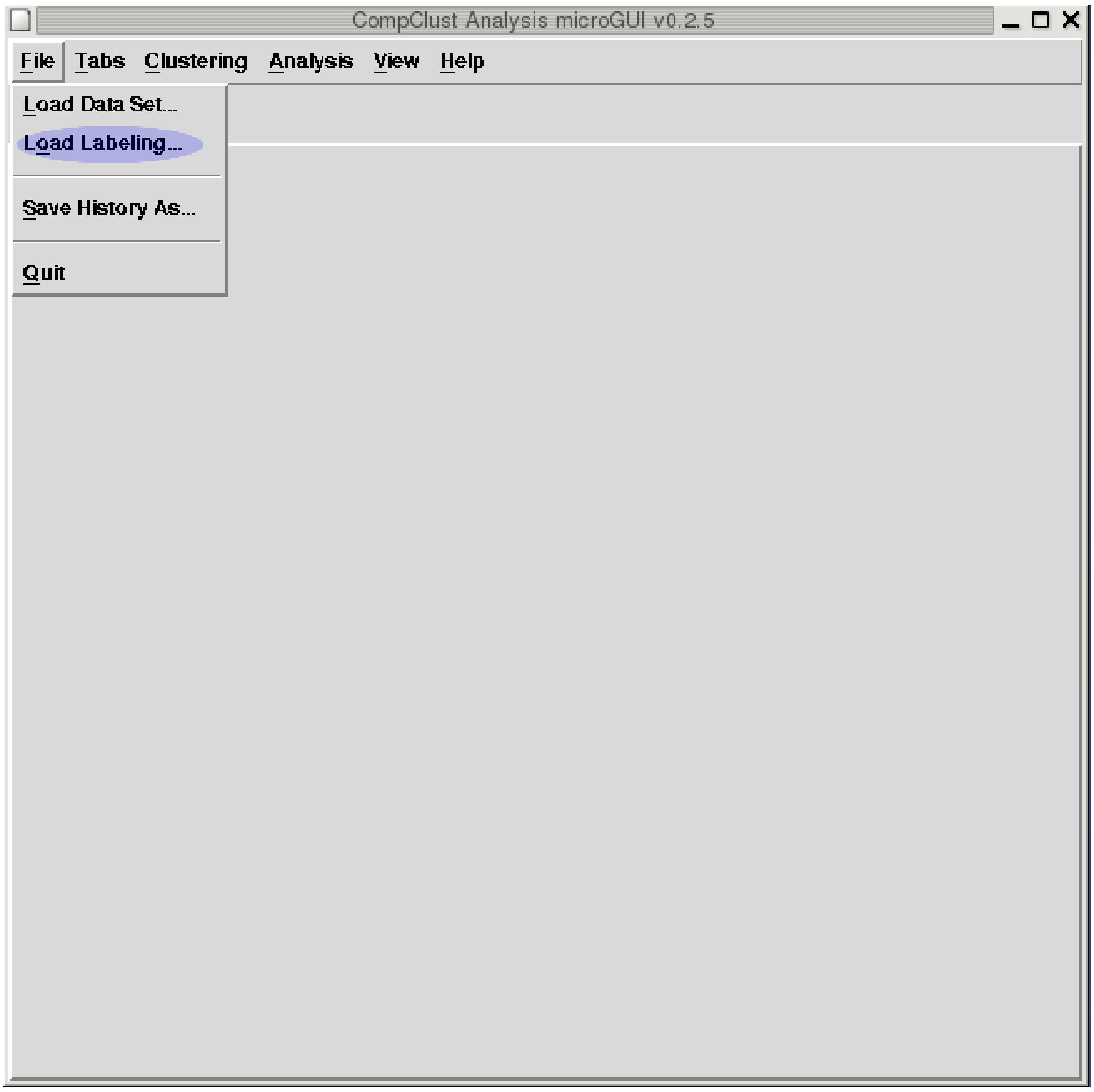


Figure 4: Load Data Set Dialog

The Load Labeling dialog box requires a little more information than the Load Data Set dialog box. Later when you are running an analysis, you will need to select one or more labelings to use, so you should choose a Labeling name which is meaningful. Also, you must tell the program whether you plan to load a row or column labeling (See section 2.3 for more information about CompClust labelings).

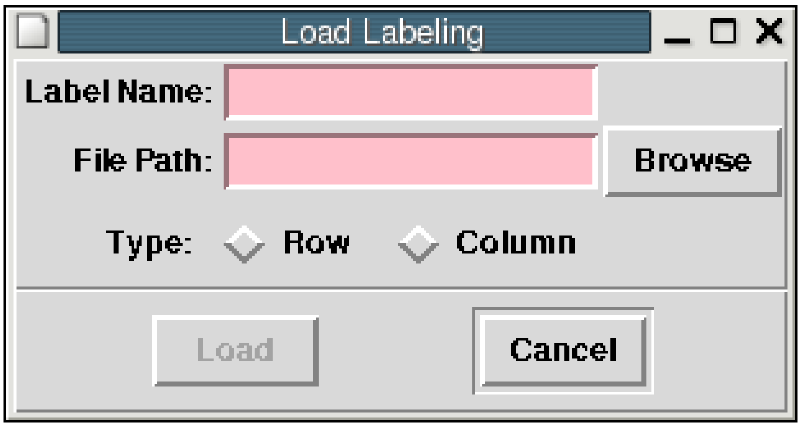
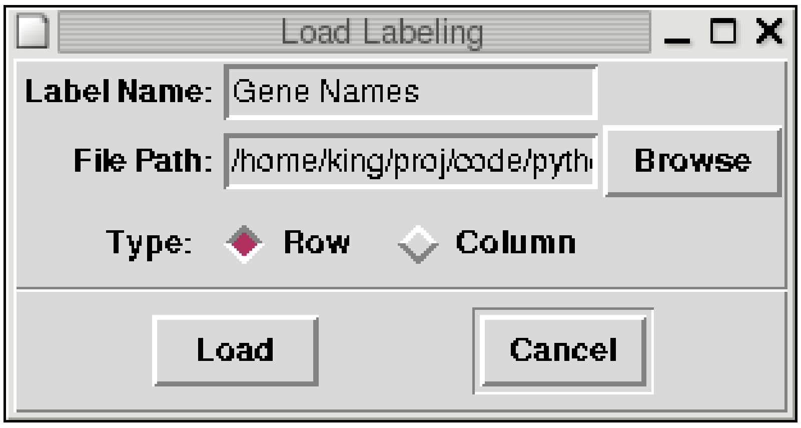


Figure 5: Load Labeling Dialog

Give your Gene Labeling a name like 'blah', press browse and select the file named 'Common-Names.rlab' and then choose the 'Row' radio button as shown below.



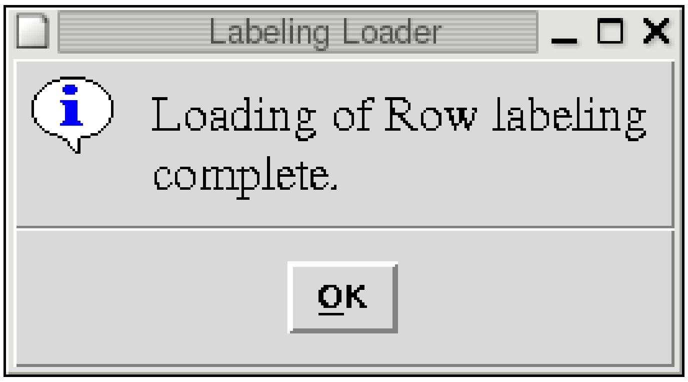
Label Name: Gene Names

File Path: /home/king/proj/code/pyth

Type: [✓ Row] Column

Figure 6: Load Labeling Dialog — Gene Names

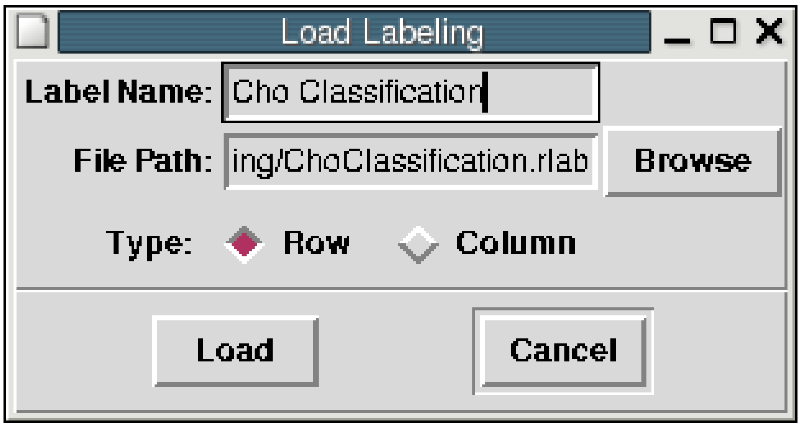
Click 'Load' when you are ready. You should see a dialog box similar to the one below if your Labeling loads successfully.



Loading of Row labeling complete.

Figure 7: Gene Name Load Labeling Complete

Later when we start analyzing our data, we will compare our clustering results (Coming up in section 3.4) to classifications made by Cho. We should attach the Labeling file 'ChoClassification.rlab' as shown below.



Label Name: Cho Classification

File Path: ing/ChoClassification.rlab

Type: [✓ Row] Column

Figure 8: Loading Cho Classification Labeling

3.4 Clustering

3.4.1 Introduction to Clustering

Clustering Algorithms are a general class of unsupervised machine learning techniques which attempt to find an approximation to the optimal partitioning (or separation) of a given data set into discrete classes or clusters. An optimal partition is is defined as the partitioning for which each data vector in a cluster is more similar to all other data vectors with the same cluster memberships than to all other data vectors with different cluster memberships. It can be shown that this problem is NP-hard (computationally intractable, if you think about it, to be certain you have found the '"true" optimal clustering you would have to search through all possible clusterings - a rapidly growing set of possibilities as the number of data points grows.).

In the context of microarray data analysis clustering has become a staple technique to provide insights into which genes have similar behavior across the conditions being assayed. Clusters essentially form groups of co-expressed genes, using these data as a starting point biologists can then start to address the more interesting questions regarding why these genes are co-expressed. For instance, are the co-regulated genes part of a similar biochemical pathway. The comparative tools provide utilities to compare clusterings, which can elucidate such things as: how different parameters affect a clustering; how different algorithms partition the data; and how different experimental perturbations affect which cluster, representing a similar expression response, a gene might belong to. This can have potentially profound effects on downstream analysis.

Although we don't stress it in this tutorial, you can cluster conditions (columns) just as easily as you can cluster genes (rows) using the same techniques.

3.4.2 DiagEM

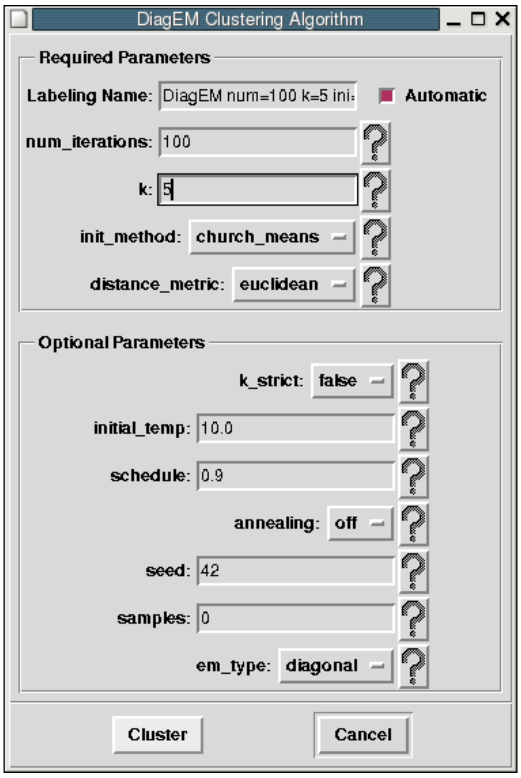
The first clustering program we will use is DiagEM. DiagEM is an implementation of the expectation maximization (EM) algorithm that attempts to fit data vectors to Gaussian clusters. The DiagEM algorithm is so named because it only uses the diagonal of the covariance matrix that describes the cluster mean and variance. Because of this the Gaussians discovered by DiagEM can only vary along the axes of the data space.

In principal the more common KMeans algorithm is a simplification of the EM algorithm which is limited to a simple high-dimensional sphere instead of the ellipsoid, or even more complex shapes, that can be described by a covariance matrix.

The EM algorithm using a full covariance matrix can create Gaussians that not only are of different widths on each dimension but can also be rotated in different direction in the data space. Unfortunately, since biological datasets frequently have many conditions this corresponds to a high dimensionality data space and because of this high dimensionality there rarely is enough data to properly estimate all of the parameters required to fill the full covariance matrix, and thus we standardized on the Diagonal EM algorithm.

We will use most of the default parameters, but we will change the number of clusters we would like DiagEM to create. Change K from two (default) to five as we will compare our results to the Cho Classification, which has been partitioned into five clusters which represent five stages of the yeast cell cycle.

Choose 'DiagEM' from the 'Clustering' menu and then change K from 2 to 5 as shown below.


DiagEM Clustering Algorithm

Required Parameters

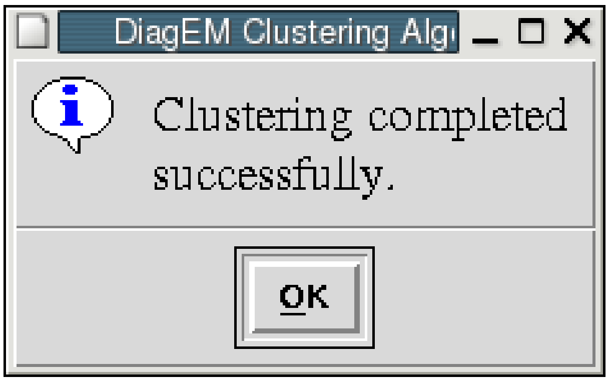
Labeling Name: DiagEM num=100 k=5 ini-
Automatic
num_iterations: 100
k: 5
init_method: church_means
distance_metric: euclidean

Optional Parameters

k_strict: false
initial_temp: 10.0
schedule: 0.9
annealing: off
seed: 42
samples: 0
em_type: diagonal

Figure 9: Clustering—DiagEM

When clustering is finished you should see a dialog box pop up similar to the one below.

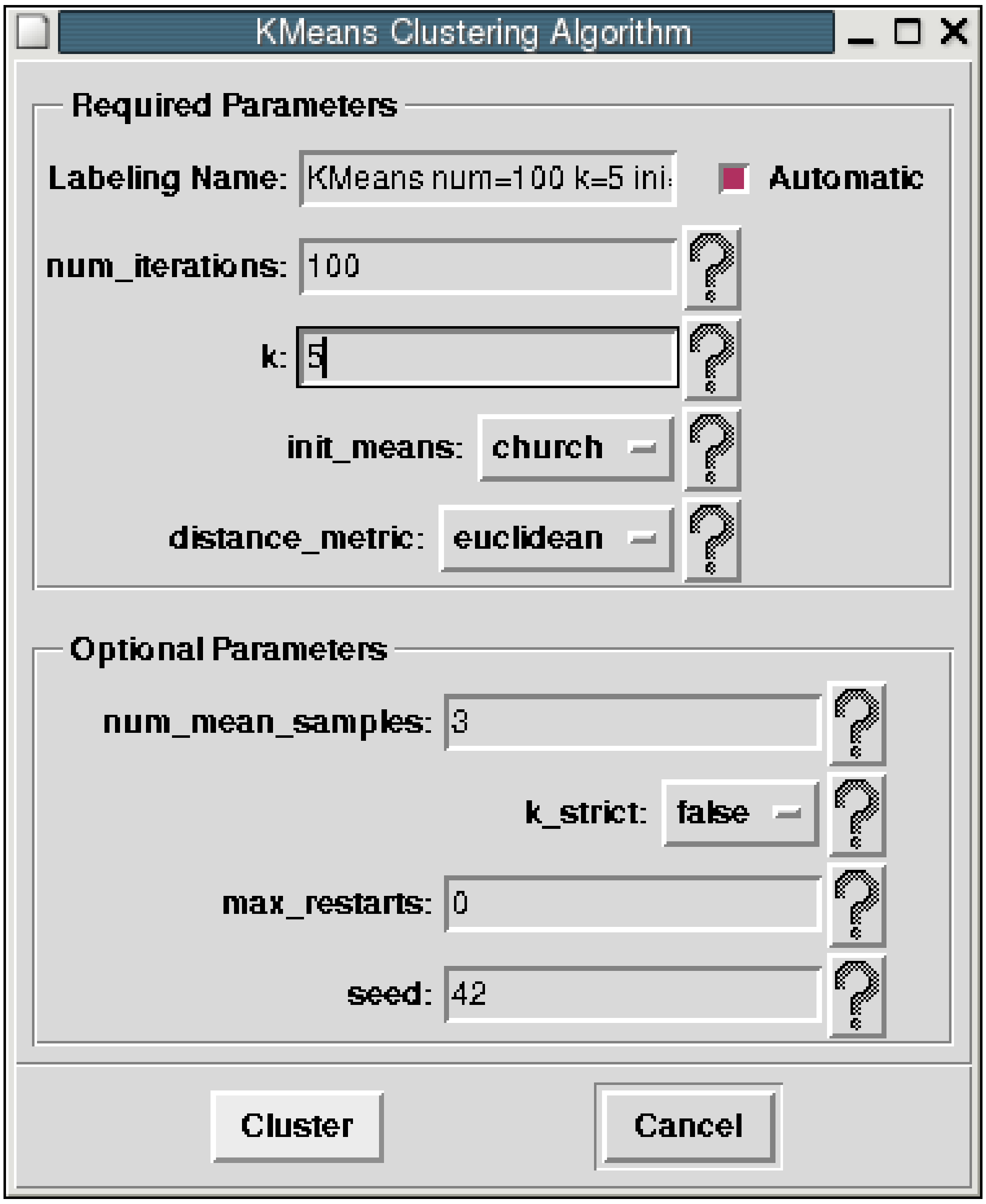

DiagEM Clustering Alg

Clustering completed successfully.

Figure 10: Clustering—DiagEM

3.4.3 KMeans

Since every clustering algorithm is different, each one may return different results. We will compare the results of KMeans with DiagEM and Cho's classifications in the analysis section of this tutorial. Select 'KMeans' from the 'Clustering' menu and then change K from 2 to 5. Click 'Cluster' to begin clustering. The KMeans dialog should be similar to the one shown below.


KMeans Clustering Algorithm

Required Parameters

Labeling Name: KMeans num=100 k=5 ini
Automatic

num_iterations: 100

k: 5

init_means: church

distance_metric: euclidean

Optional Parameters

num_mean_samples: 3

k_strict: false

max_restarts: 0

seed: 42

Figure 11: Clustering—KMeans

3.5 Analysis

3.5.1 Introduction

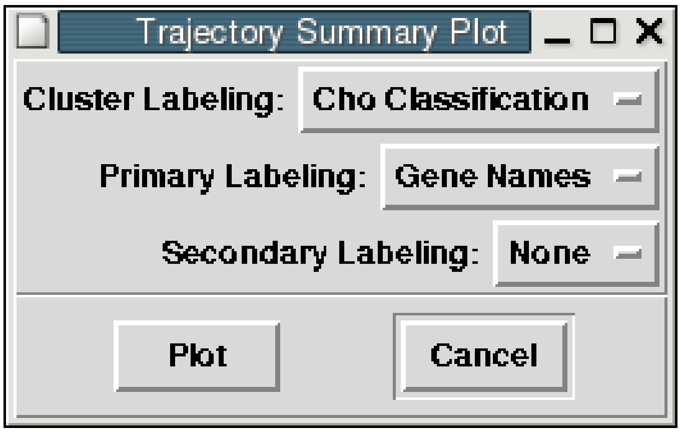
Okay, so we have labelings, both ones that we have loaded, but also labelings created by DiagEM and KMeans. In section 3.5 we will explore the data and view/compare the results of the clustering algorithms. This will be done with a visualization tool called IPlot, written by Chris Hart, which was built on top of pyMLX, written by Ben Bornstein, Chris Hart, Lucas Scharenbroich, and Diane Trout.

At any time if you feel you have too many Tabs open, or you are done with a plot, select 'Close Current Tab' from the 'Tabs' menu.

3.5.2 Trajectory Summary

The Trajectory Summary is a great tool for viewing your data. Given a Cluster Labeling and a Gene Labeling, this tool will allow you to easily visualize and explore your data set.

Let's start by choosing 'Trajectory Summary' from the 'Analysis' menu. To start, select 'Cho Classification' for the 'Cluster Labeling'. So that we know which row (vector) represents which Gene, choose 'Gene Names' for 'Primary Labeling'. The 'Primary Labeling' is the labeling which is displayed when you click on an individual vector. Click 'Plot' when you are ready to view the Trajectory Summary.


Trajectory Summary Plot

Cluster Labeling: Cho Classification

Primary Labeling: Gene Names

Secondary Labeling: None

Figure 12: Clustering—KMeans

Your 'Cho Classification' Trajectory Summary should look similar to the image below. You should have five clusters separating the gene expression data into five stages of the yeast cell cycle. The blue lines are the mean trajectory for a given cluster. The red lines are the standard deviation from the mean.

This is a helpful view to get an idea of what your clusters are doing, but if you want a more detailed view, left click on the plot. In this case, lets look at the 'Late G1' cluster. Click on the plot for the 'Late G1' cluster now.

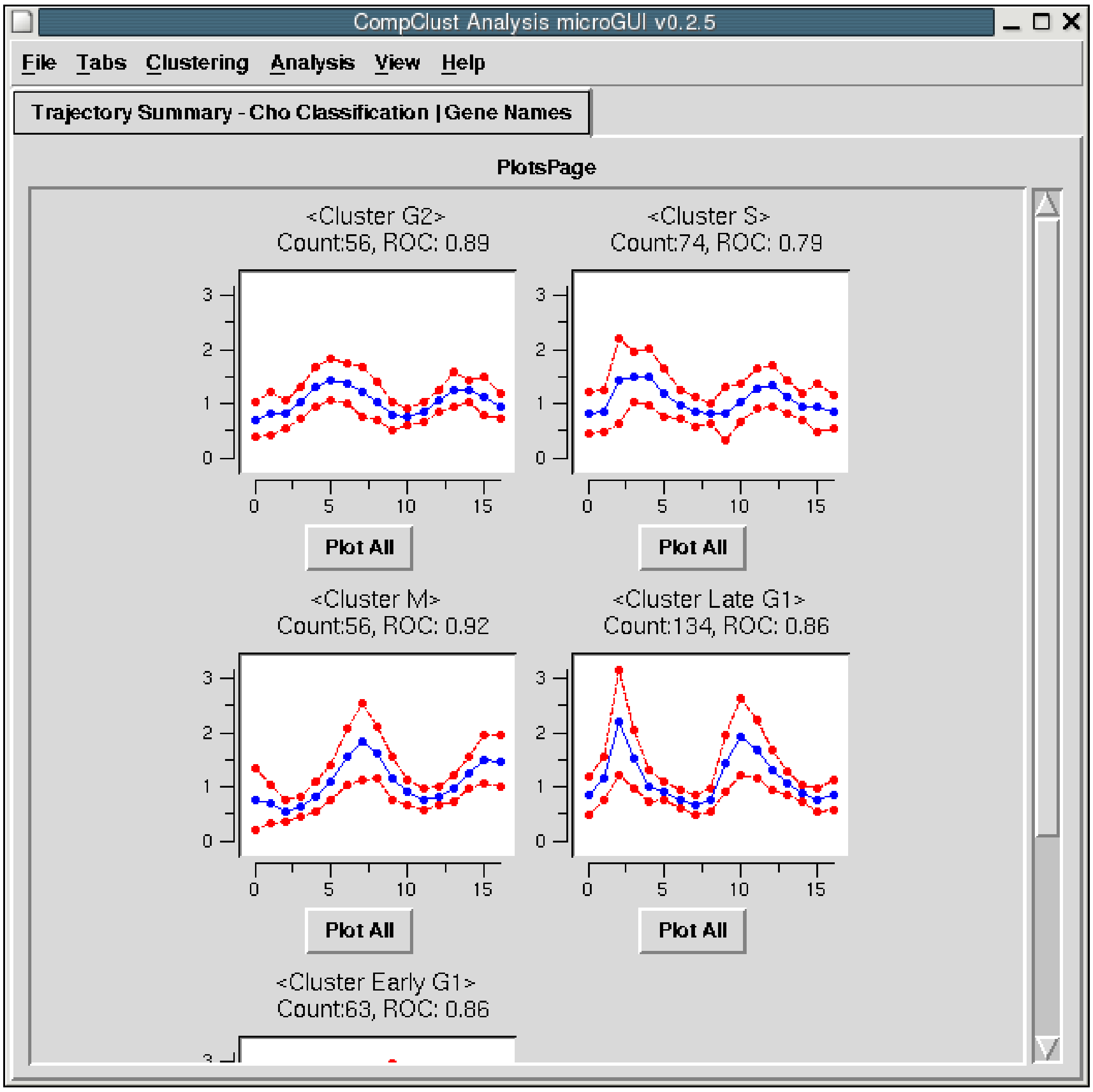


Figure 13: Clustering—KMeans

You should get a plot that looks like the following. The coloring of the trajectories is based on the expression level at time 0 by default. In a future version we may expose the ability to easily change the coloring schema within the GUI itself, but for now if you have the desire to change the coloring, you will have to use the 'Analysis Shell' which is discussed in section 3.6.

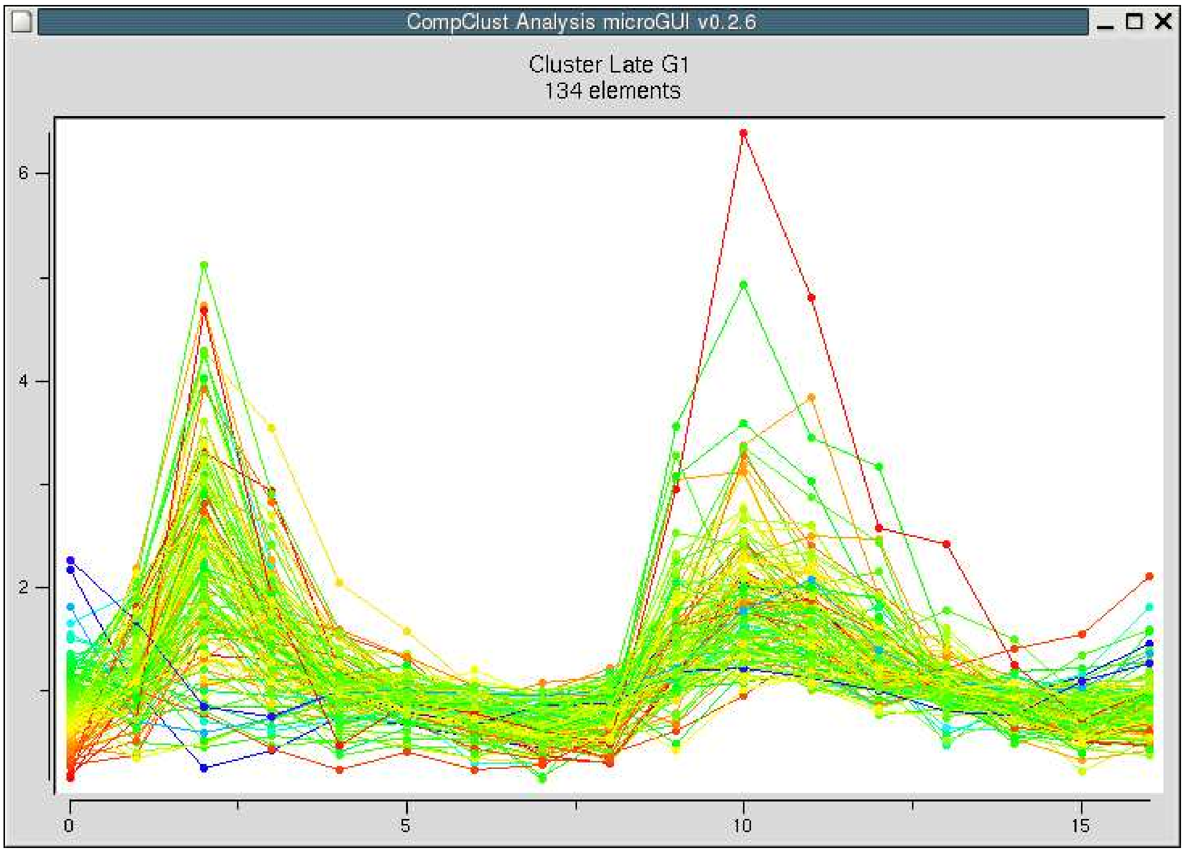


Figure 14: Plot All - Late G1

If you click on any point in the plot, a box in the top right will show up with the text 'Gene Name: (x-cord, y-cord)'. Click on the point shown in the diagram below and you should see 'SCW11: (10.00, 4.92)'. If you control click on the point shown below, a new dialog box appears showing the trajectory for 'SCW11' as shown in Figure 16 on the next page.

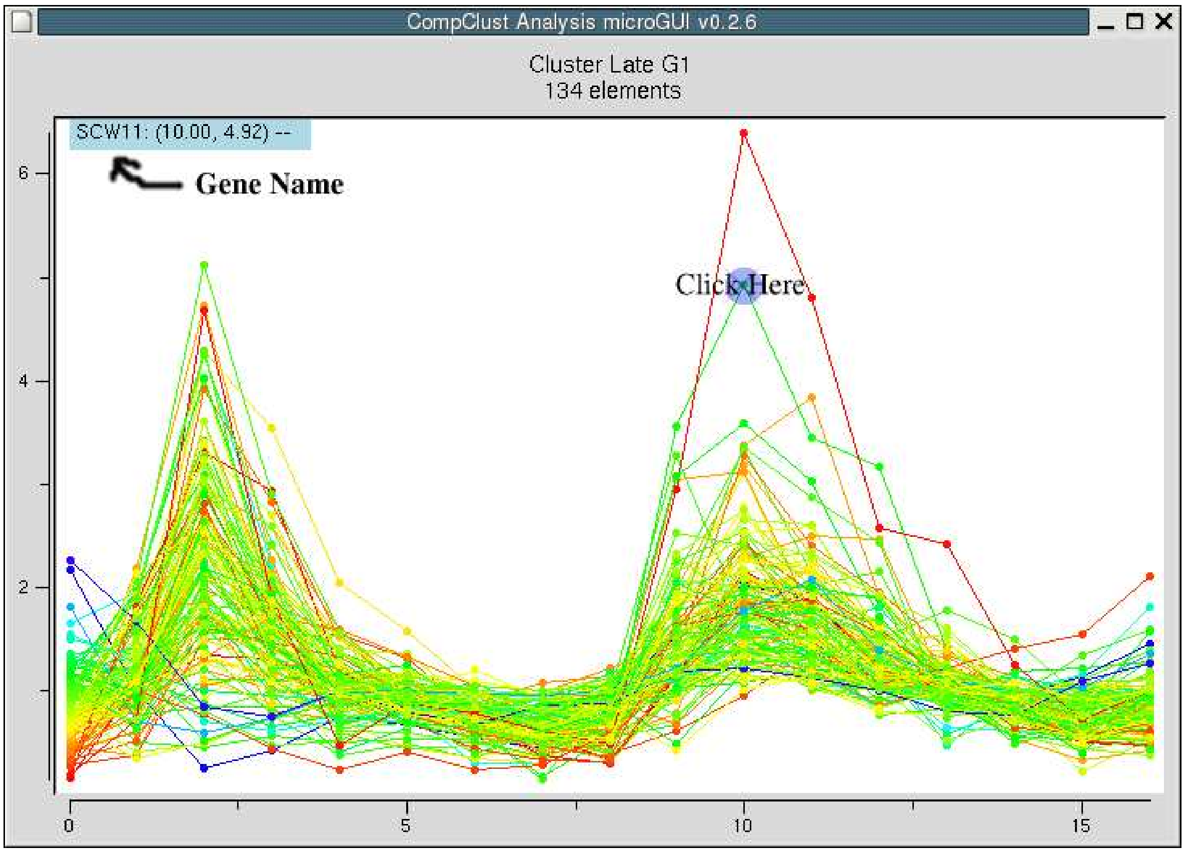
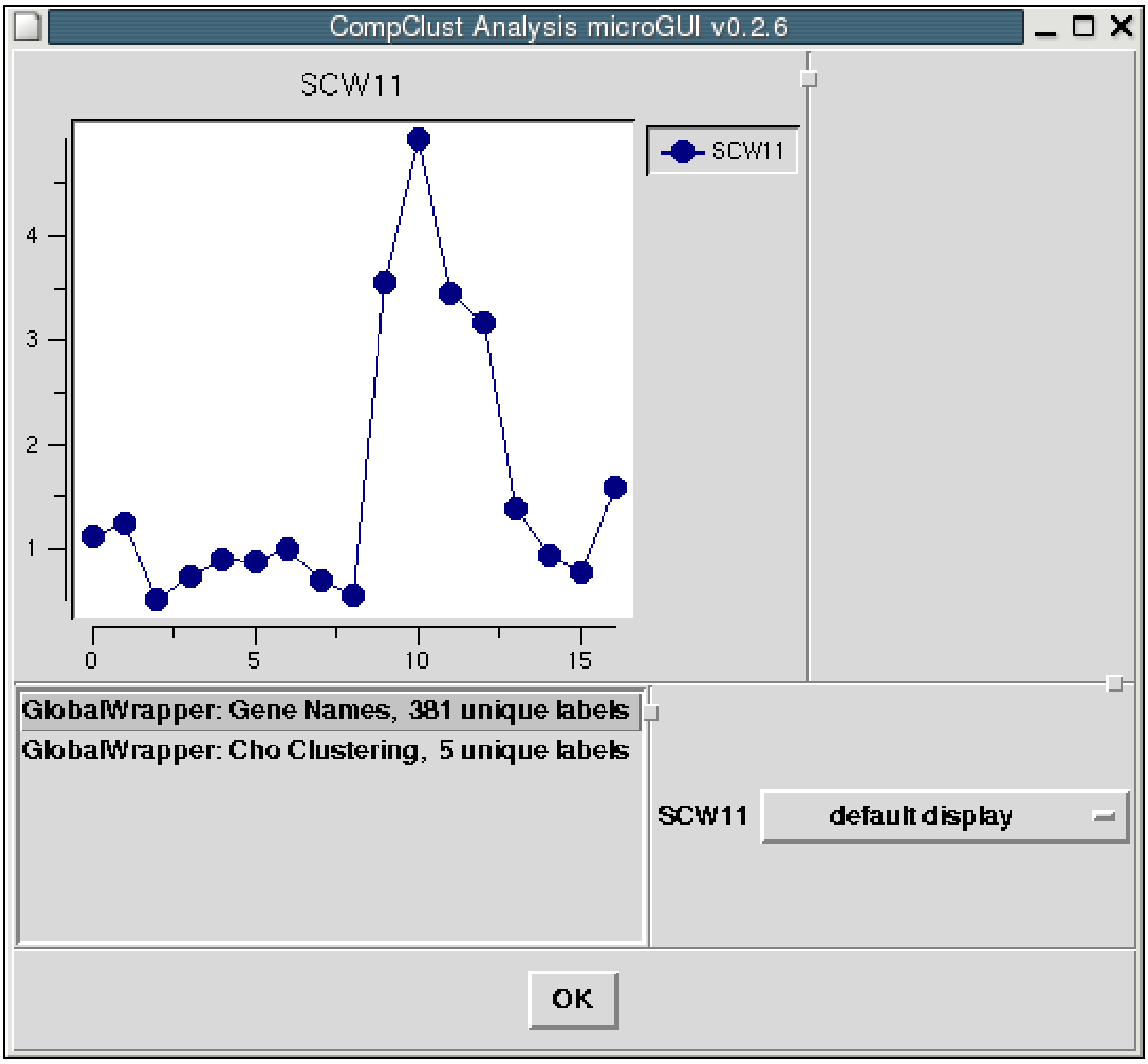


Figure 15: Plot All - Late G1 - SCW11

Note that at the bottom of the figure below, there are two Labelings, 'Gene Names' and 'Cho Classification'. If you click on the 'Gene Names' labeling you should see a pull down menu show up in the bottom right. If you then change this from 'default display' to 'Highlight This Group', this gene will be highlighted in your 'Late G1 - Plot All' display as shown in Figure 18 on the next page.



GlobalWrapper: Gene Names, 381 unique labels
GlobalWrapper: Cho Clustering, 5 unique labels

SCW11 default display

Figure 16: Plot All - Late G1 - SCW11

At this point you can 'Ctrl + Click' on other gene vectors and highlight them as well. This is all we will cover on the Trajectory Summary Plot for this tutorial. Feel free to explore more on your own or continue on to the next section of the tutorial.

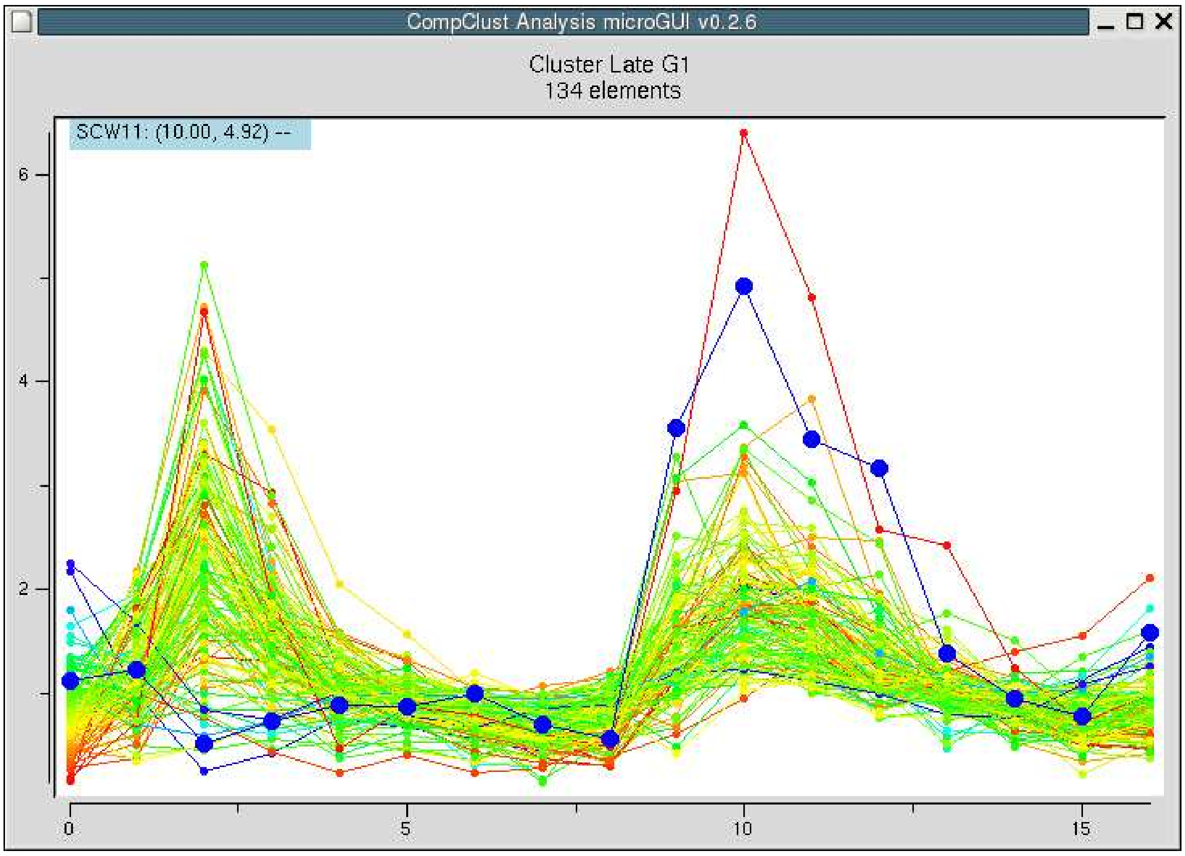
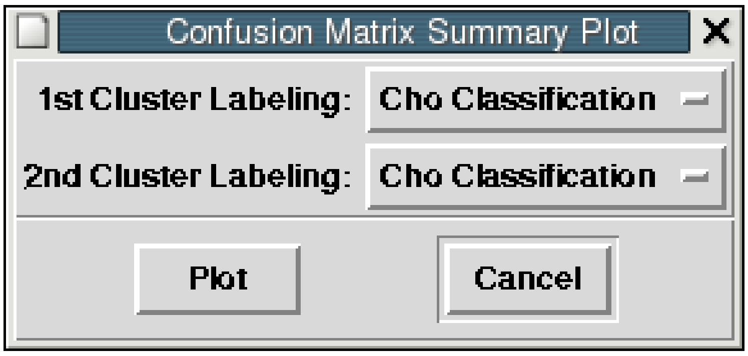


Figure 17: Plot All - Late G1 - SCW11 Highlighted

3.5.3 Confusion Matrix

Now that we have an idea of what our clusters look like from the Trajectory Summary Plots, we will compare the 'Cho Classification', 'DiagEM w/ K=5', and 'K-means w/ K=5'. This will be done using a Confusion Matrix Plot.

For starters, let's compare the 'Cho Classification'[1] to itself. Select 'Build Confusion Matrix' from the 'Analysis' menu. Then select 'Cho Classification' for the '1st Clustering Labeling' and '2nd Clustering Labeling' as shown in the figure below. Click 'Plot' when you are ready to move on.



Confusion Matrix Summary Plot

1st Cluster Labeling: Cho Classification

2nd Cluster Labeling: Cho Classification

Figure 18: Confusion Matrix Dialog - Cho vs. Cho

[1]See section 3.3 for information on loading labelings.

You should get a Confusion Matrix plot similar to the following figure. Notice that there are two 'Trajectory Summary' sections being displayed with white backgrounds (top row and last column). Each one of these sections is a clustering, in this case 'Cho Classification' versus itself. If you look at the five clusters in the top row, you'll notice that I have super-imposed green and red bars in the figure below. The green bars are highlighting the number of genes[2] in a given cluster. The red bars are highlighting the name of the clustering.

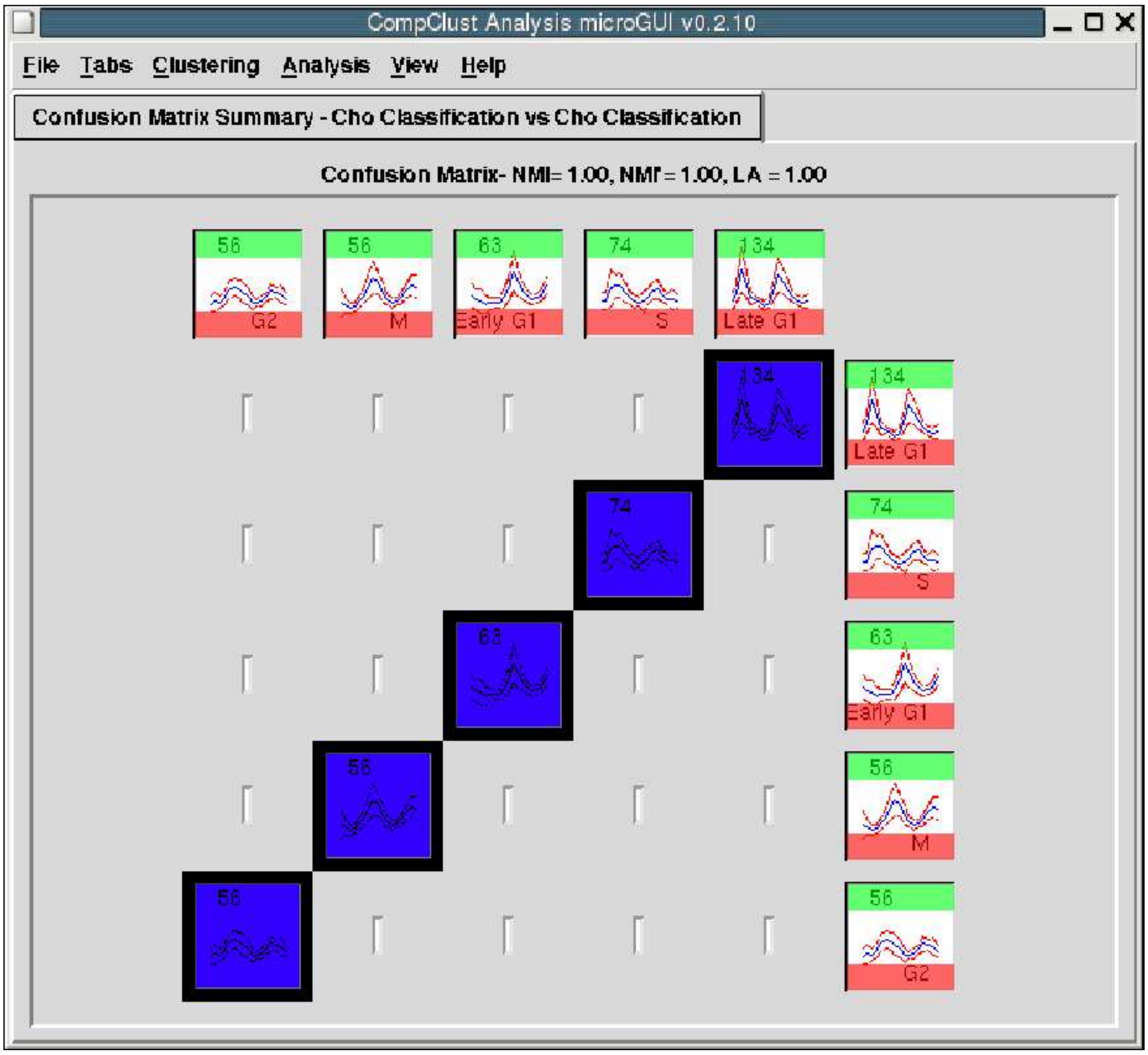

CompClust Analysis microGUI v0.2.10
File Tabs Clustering Analysis View Help
Confusion Matrix Summary - Cho Classification vs Cho Classification
Confusion Matrix- NMI= 1.00, NMI'= 1.00, LA = 1.00
56
G2
56
M
63
Early G1
74
S
134
Late G1

Figure 19: Confusion Matrix Tab - Cho vs. Cho

[2]CompClust is capable of supporting other types of data beyond gene expression data.

What is this matrix telling us? It's showing us the number of members of column Y that are showing up in row X. For example, if we look at column 2 (M Phase Cluster) and compare it to row 2 (S Phase Cluster), we see that the 'M Phase Cluster' has no members that are shared with the 'S Phase Cluster' (see figure below). Later when we compare our clustering results to Cho's, things won't be as clear as this. If you look at row 1 (Late G1) vs column 5 (Late G1) you'll see that 134 out of 134 members are shared between the two clusters (because they are the same cluster).

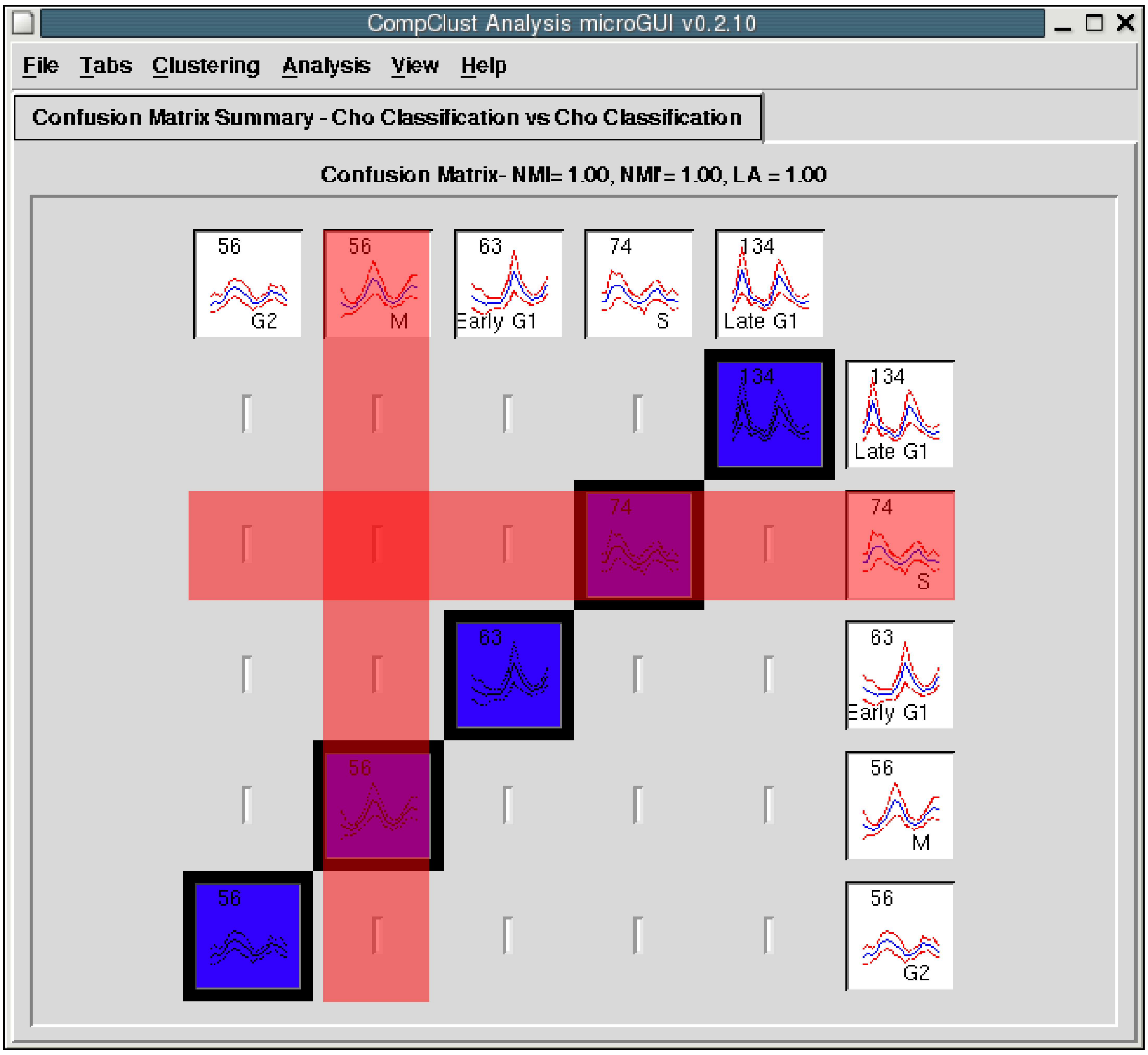
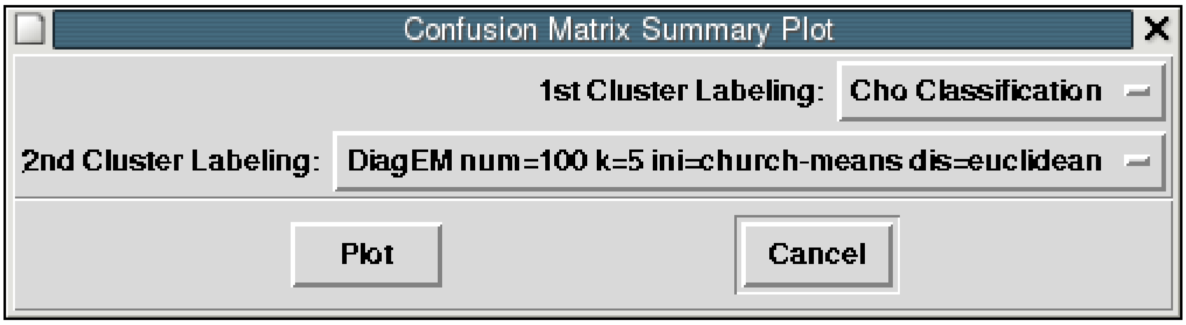

CompClust Analysis microGUI v0.2.10
File
Tabs
Clustering
Analysis
View
Help
Confusion Matrix Summary - Cho Classification vs Cho Classification
Confusion Matrix- NMI= 1.00, NMI'= 1.00, LA = 1.00
56 G2
56 M
63 Early G1
74 S
134 Late G1
134
134 Late G1
74
74 S
63
63 Early G1
56
56 M
56
56 G2

Figure 20: Confusion Matrix Tab - Cho vs. Cho

Now let's move onto a more interesting comparison. Let's compare the 'Cho Classification' to our clustering of 'DiagEM w/ K=5'[3]. Select 'Build Confusion Matrix' from the 'Analysis' menu. Then select 'Cho Classification' for the '1st Clustering Labeling' and 'DiamEM...k=5...' for the '2nd Clustering Labeling' as shown in the figure below. Click 'Plot' when you are ready to move on.


Confusion Matrix Summary Plot

1st Cluster Labeling: Cho Classification

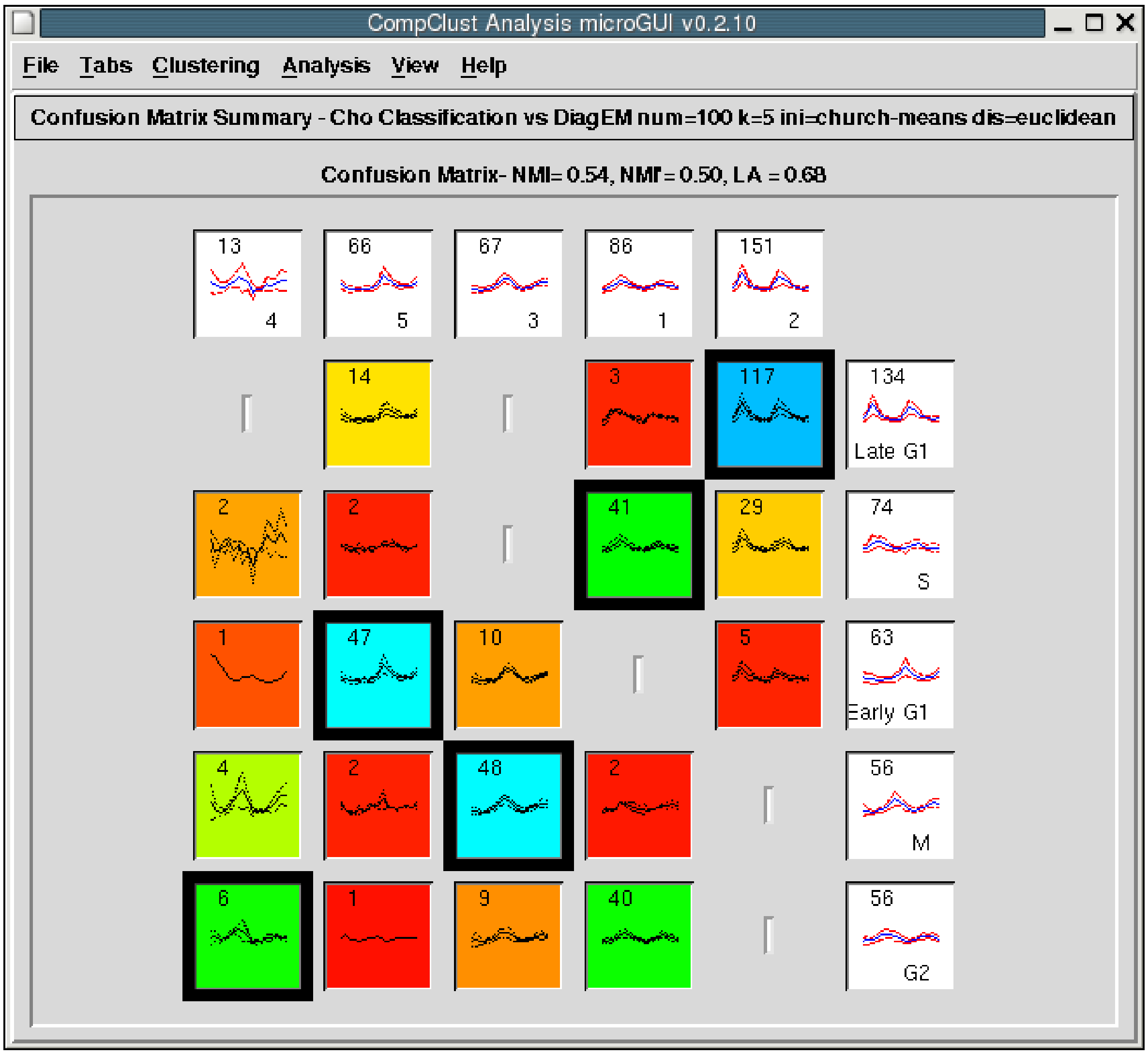
2nd Cluster Labeling: DiagEM num=100 k=5 ini=church-means dis=euclidean

Figure 21: Confusion Matrix Dialog - Cho vs. DiagEM

[3]See section 3.4 if you haven't run DiagEM yet.

As you can see from the figure below, you get a much more interesting plot. We have 'DiagEM' on the X axis and 'Cho Classification' on the Y axis. Note that DiagEM gave the clusters numeric labelings as it had no way of knowing any biologically related information for naming purposes. It is the job of the user to try to figure the meaning of those cluster labels.

I'll start by describing the coloring scheme. The colors range from red to blue, where pure red is 0% of the members for a given row and column are shared by the two clusters. Pure blue would mean that 100% of the members are shared between the two clusters. If we look at column 1 (DiagEM Cluster #4) and compare it to row 2 (Cho's S Phase Cluster) we see that 2 out of 13 (15.38%) members are shared between the two clusters, which is why it has an orange color. [4]



Confusion Matrix Summary - Cho Classification vs DiagEM num=100 k=5 ini=church-means dis=euclidean

Confusion Matrix- NMI= 0.54, NMI' = 0.50, LA = 0.68

	13 4	66 5	67 3	86 1	151 2
134 Late G1		14		3	117
74 S	2	2		41	29
63 Early G1	1	47	10		5
56 M	4	2	48	2	
56 G2	6	1	9	40	

Figure 22: Confusion Matrix Tab - Cho vs. DiagEM

[4]Note that all color calculations are based on all the comparisons for a given column. If you were to redo this plot as 'DiagEM' vs 'Cho' instead of 'Cho' vs 'DiagEM', the results will probably be very similar, but the color coding may change significantly. If we looked at the same comparison of 'DiagEM Cluster #4' vs 'Cho's S Phase Cluster' in the 'DiageEM' vs 'Cho' plot, the color would be calculated as 2 out of 74 (2.70%), which would make it much more red than it is in our current plot.

If you look at column 5 (DiagEM Cluster #2) vs row 1 (Late G1) on the figure on the previous page, you will notice that 117 members out of 151 are shared between the two clusters.

3.5.4 ROC Analysis

ROC Analysis is a tool to examine the overlap of the members of a cluster with that of the surrounding space. It does this by comparing the false positive rate on the X axis and the false negative rate on the Y axis.

To visualize the way that the ROC curve is computed imagine a hypersphere that starts with zero size at the cluster center, then as the sphere grows for every point you hit that is in the cluster you increment along the Y axis, for every point that you hit that is not in the cluster you increment along the X axis. A perfect ROC score would have a vertical line at x=0, followed by a horizontal line at y=1.

The way that CompClustTk visualizes this shows the standard ROC curve along the left, with histograms of the cluster members and non-members on the right.

To visualize an ROC curve go to the Analysis menu and select Cluster ROC analysis.

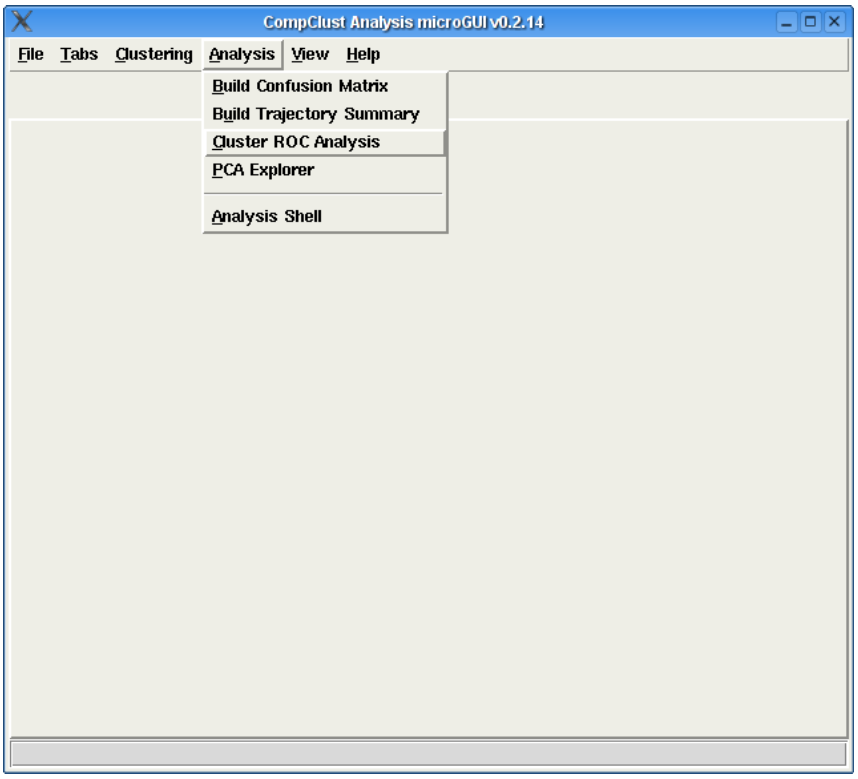
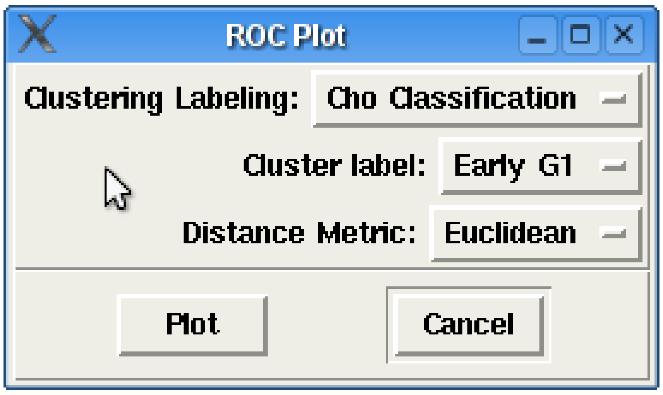


Figure 23: Cluster ROC Analysis

The menu item will bring up a dialog box that allows you to select which Cluster Labeling to use. In this case let's select the Early G1 cluster from the Cho Classification.

The Clustering Labeling specifies which clustering that one wants to explore, while Cluster label allows one to specify which cluster you want to consider the "inside" cluster for the analysis.



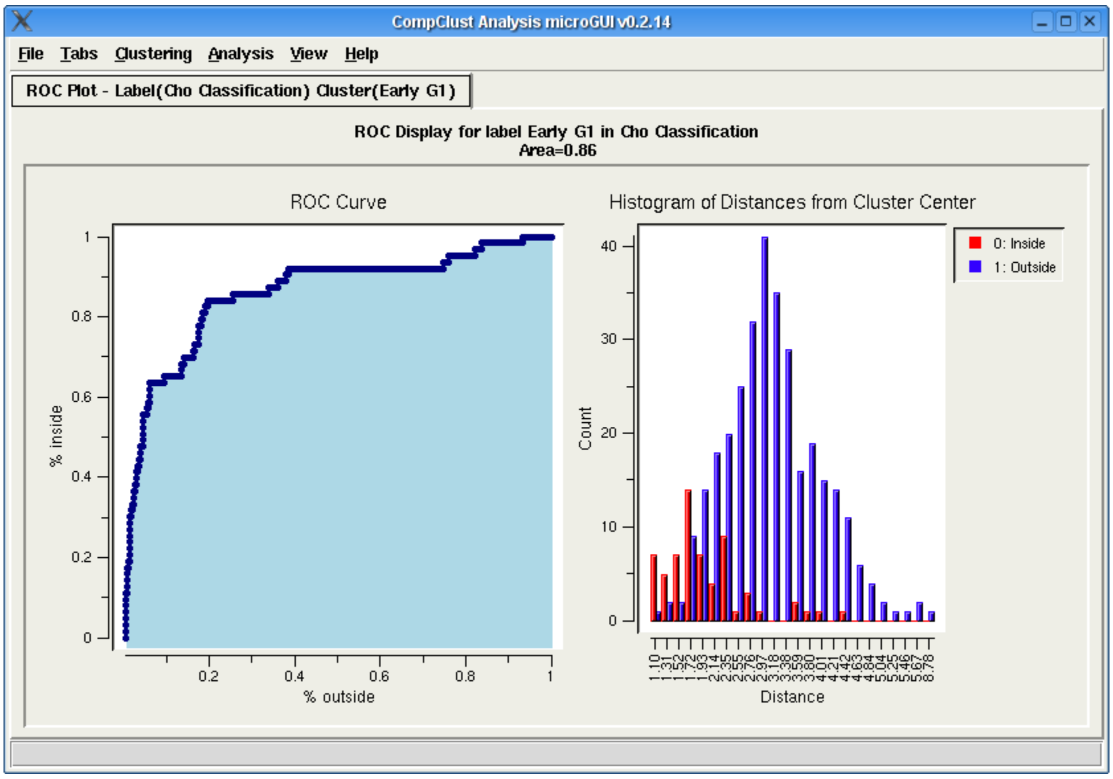
Clustering Labeling: Cho Classification
Cluster label: Early G1
Distance Metric: Euclidean

Figure 24: ROC Analysis Dialog Box

Once selected you can see the comparisons, on the left is the standard ROC curve, and on the right is the plot of how many points were found at each distance bin, red represents cluster members, and blue represents everything else.

The more separated those two histograms are the better the clustering.

This is not a particularly well separated cluster. If one looks at the histogram of distances, there are several data points that are considered part of this cluster that are actually quite far from the cluster center.



CompClust Analysis microGUI v0.2.14
File Tabs Clustering Analysis View Help
ROC Plot - Label(Cho Classification) Cluster(Early G1)
ROC Display for label Early G1 in Cho Classification
Area=0.86
ROC Curve
% inside
% outside
0
0.2
0.4
0.6
0.8
1
Histogram of Distances from Cluster Center
Count
0
10
20
30
40
Distance
1.10 1.31 1.52 1.72 1.93 2.14 2.35 2.55 2.76 2.97 3.18 3.38 3.59 3.80 4.01 4.21 4.42 4.63 4.84 5.04 5.25 5.46 5.67 8.78
🟥 0: Inside
🟦 1: Outside

Figure 25: ROC Analysis Cho Classification of Early G1

In this case we look at one of the better clusters found by our EM algorithm. You can see in the histogram that the elements in the cluster rapidly taper off as one moves from the cluster center.

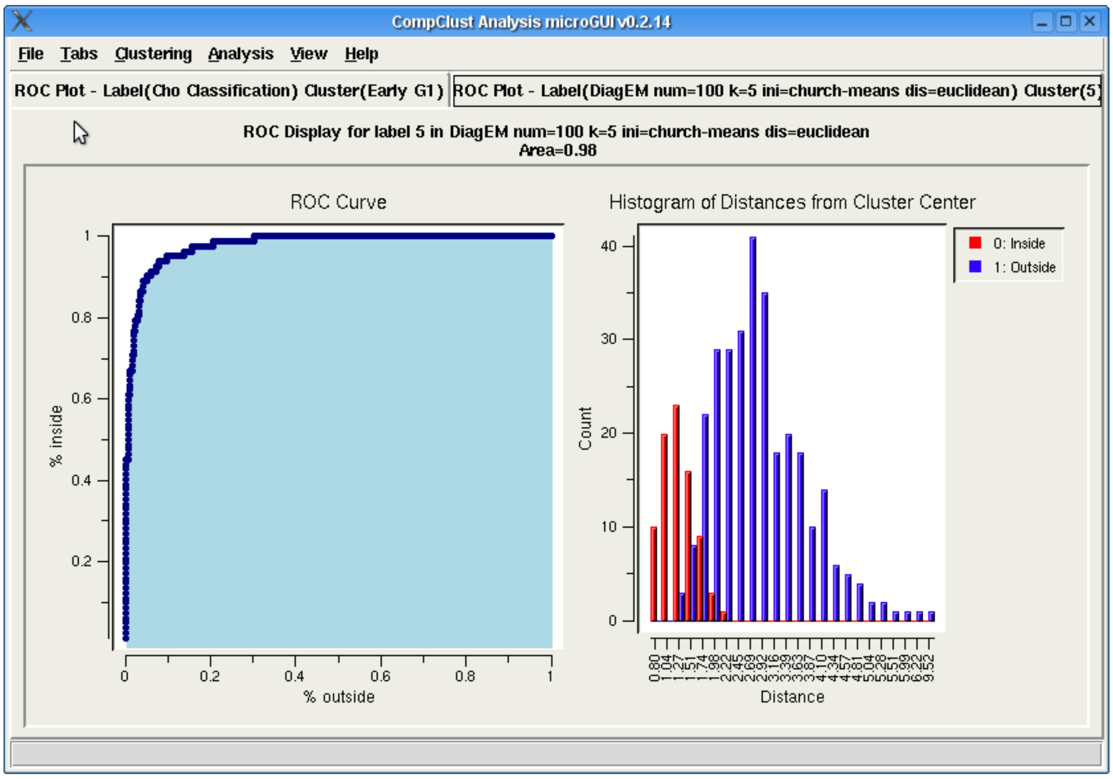

CompClust Analysis microGUI v0.2.14
File Tabs Clustering Analysis View Help
ROC Plot - Label(Cho Classification) Cluster(Early G1)
ROC Plot - Label(DiagEM num=100 k=5 ini=church-means dis=euclidean) Cluster(5)
ROC Display for label 5 in DiagEM num=100 k=5 ini=church-means dis=euclidean
Area=0.98
ROC Curve
% inside
1
0.8
0.6
0.4
0.2
0
0.2
0.4
0.6
0.8
1
% outside
Histogram of Distances from Cluster Center
Count
40
30
20
10
0
🟥 0: Inside
🟦 1: Outside
0.80 1.04 1.27 1.51 1.74 1.98 2.22 2.45 2.69 2.92 3.16 3.39 3.63 3.87 4.10 4.34 4.57 4.81 5.04 5.28 5.51 5.99 6.22 6.52
Distance

Figure 26: ROC Analysis DiagEM of cluster 5

To be fair, the EM algorithm we use is building clusters using a Gaussian cloud, which does a very good job of

3.5.5 PCA Explorer

To Be Written

3.6 Advanced Analysis - IPython

3.6.1 Introduction

The “Analysis Shell” is basically just an IPython[5] command prompt which allows you to use the Python[6] programming language. This also gives you access to all of the CompClust Python package and the GUI internals. What this means is that if the GUI does NOT currently do something you would like it to, you can probably make it happen using Python in the “Analysis Shell”.

To get you started with Python, we recommend reading the Python tutorial at http://docs.python.org/tut/tut.html.

[5]IPython - http://ipython.scipy.org/
[6]Python - http://www.python.org/

3.6.2 Analysis Shell and Log 2 Transform Example

One common thing you may wish to do which currently is not supported in CompClustTkis the ability to transform your data set. The CompClust Python package supports this extensively among many other powerful features. In this case, let's say you want to log2 transform your data set. Load a data set as shown in section 3.2 and then launch the 'Analysis Shell' from by going to the 'Analysis' menu and choosing 'Analysis Shell'. You will find the 'Analysis Shell' in the original shell window you used to launch CompClustTkor in the 2nd window that was launch upon starting CompClustTk.

To give you an glimps of the many powerful things one can do with the CompClust Python package, I'm going to show you how to do a log2 transform of the dataset, but I'm going to do it by using a data set 'View' called the 'FunctionView'. The 'FunctionView' allows you to transform your data set by passing in a function which will be applied to every element of your data set. The function you pass to the 'FunctionView' needs to take one argument and return one value. In our case, we will make a function which will take on element of the data set, convert it to log2, and return. As you can imagine, any function you can come up with can be applied using this method.

The 'FunctionView' is only one of the many 'Views' one can use on your dataset. The nice thing about a 'View' is that it doesn't actually store a copy of the data set. It gets the data from the original dataset when you access the 'View'. This also means that any 'Labelings' you have attached to your original data set will also be accessable by your 'View', even if your 'View' is only a subset of your original data set. Since a 'View' implements all the functions a data set object has, it's usable where ever a function asks for a data set. This also means you can create another view from a view. Don't worry if that didn't make much sense, basically what it means is that it's relatively memory effecient and easy to use (from a programmer's point of view).

If you would like to see what views are available type the following from the analysis shell and then press TAB after typing the period:

```
views.
```

In the case that you are using the CompClust Python package from within Python itself without a GUI, then you will need to import the views module by typing the following command. If your using the 'Analysis Shell' then the following command has already been executed for you.

```
from compClust.mlx import views
```

To get information on any particular view, or for any Python object for that matter, type the variable/function/object name, then a '?' and press enter. For example for the 'FunctionView' you would type:

```
views.FunctionView?<press-enter>
```

Now onto example. The first thing we are going to do is create the log2 function we are going to pass to the 'FunctionView'. To do this, we will need to load the 'math' Python module by typing:

```
import math
```

We are going to use the math.log function, which takes two arguments, number and base. But the 'FunctionView' expects to receive a function which is takes only one argument, we need wrap the math.log function with the 'base' argument set to 2. Type the following do define the log2 function:

```
def log2(x):
  return math.log(x, 2)
```

Note that you need to press enter twice after writting the last line of the function. This tells Python to go ahead and define the function. If everything went well, your command prompt should look like this:

```
In [5]:def log2(x):
   ...:    return math.log(x, 2)
   ...:

In [6]:
```

If you write more Python code on the next line rather than pressing enter twice, you'll probably end up with a SyntaxError like the following:

```
In [5]:def log2(x):
   ...:    return math.log(x, 2)
   ...:log2(2)
------------------------------------------------------------
   File "<console>", line 3
     log2(2)
        ^
SyntaxError: invalid syntax
```

Feel free to try out your new function by typing:

```
In [9]:log2(2)
Out[9]:1.0
```

Now that we have our function, it's time to get the data set which you've already loaded from within the GUI. To grab the data set, type the following:

```
dataSet = gui.data['myDataSet']
```

Since we are going to replace gui.data['myDataSet'] with the transformed data set, if you want access to the original dataSet, you should save the data set to a variable you can access later. If you save it to the gui.data Python dictionary, then you will be able to access the original data set even if you close the 'Analysis Shell' and re-open it later. To do this type the following:

```
gui.data['originalDataSet'] = dataSet
```

Now we will create the log 2 transformed view (a.k.a. data set). To do this, call the 'FunctionView' with the data set and the log2 function by typing:

```
log2DataSet = views.FunctionView(dataSet, log2)
```

Now that you have the log 2 view of your data, if you want to be able to view it in CompClustTk, you will need to store it in gui.data['myDataSet'] so that the GUI knows that you want it to use the log 2 view when doing visualizations. To do this type the following:

```
gui.data['myDataSet'] = log2DataSet
```

That's it, now you can go back to the GUI and use your log 2 transformed data. At this point you can either quit out of the 'Analysis Shell' or leave it open; it's up to you. Note that if you close the 'Analysis Shell', you will be able to launch it again, but all of your local variables such as your log2 function will be lost.

If you don't want to lose what you've written, or you want to add a lot of code all at once, IPython (a.k.a Analysis Shell) will allow you to use a text editor to write your code. To launch the default text editor, type the following where ¡path¿ is that path to the file you want to create/use.

```
edit <path>.py
```

That command should launch a text editor for you to use. Type your Python code and when your done, save the file and exit out of the text editor. IPython will then read in and execute your Python code. If you get some sort of error or you want to make a change, just type the same with command as before and you will be able to modify the code some more.

Using the edit command to load in and write Python code will allow to to quickly load, test, and edit your Python code. This can be used to do automated loading of data/labelings or some advanced analysis and then view your results from within the GUI.

By the way, to change the default editor, set the environment variable 'EDITOR' to the name of or the path to the editor you wish to use.

3.6.3 Advanced Analysis Shell

At some point if you want to make plots appear within the gui, but doing so from the analysis shell, there are a few things you should know. When making plots or adding your on Tkinter code from within the 'Analysis Shell', you may get strange errors like 'blt::graph' won't make much sence. This happens if you try to create a new root Tkinter object when one already exists. The root Tkinter object can be found at 'gui.parent'. If you want to create a new 'Pop Up' style window, type the following from the 'Analysis Shell':

```
toplevel = Tkinter.Toplevel(master=gui.parent)
```

This 'toplevel' object can be passed to just about any IPlot visualization to become the parent window for that Plot or it can be used as the parent window for new Tkinter widgets.

If you want new plots or new Tkinter widgets to show up in a tab in the CompClustTk GUI, then you need to create a new page and use that variable instead of a root Tkinter object or a 'toplevel' object. Type the following to create the new 'page' object and then tell Python to automatically select this new page for you:

```
page = gui.notebook.add('My New Tab')
gui.notebook.selectpage('My New Tab')
```

If this doesn't make much sense and/or if your now interested in learning to make GUIs using Python and Tkinter, check out: http://www.pythonware.com/library/tkinter/introduction/

3.6.4 CompClust Python Package

If your interested in learning more about what you can do with the CompClust Python Package, check out the tutorials in the next section. When using the CompClust Python packacge the analysis posibilities are only limited by your imagination ... Well, that and CPU power and RAM, your understanding of Python and CompClust Python package. Well, at least you don't have to be limited by what the GUI can do.

4 More of CompClust

4.1 Other CompClust Tutorials

The following tutorials can be found at http://woldlab.caltech.edu/compClust/.

4.1.1 A Quick Start Guide to Microarray Analysis using CompClust

"A Quick Start Guide to Microarray Analysis using CompClust" written by Christopher Hart, covers how to use the Python CompClust environment to do microarray analysis. It may give you a better understanding of the IPlot tools (Trajectory Summary, Confusion Matrices, etc.). It will also teach you how to use some of the more advanced features of CompClust which haven't been exposed to CompClustTk.

4.1.2 A First Tutorial on the MLX schema

"A First Tutorial on the MLX schema" written by Lucas Scharenbroich, covers the MLX schema. If your want use the full power of compClust using python.

5 Acknowledgements

References

Cho, R. J., Campbell, M. J., Winzeler, E. A., Steinmetz, L., Conway, A., Wodicka, L., Wolfsberg, T. G., Gabrielian, A. E., Landsman, D., Lockhart, D. J., and Davis, R. W. (1998). A genome-wide transcriptional analysis of the mitotic cell cycle. *Mol Cell*, 2(1):65–73. (eng).

Hart, C. E., Sharenbroich, L., Bornstein, B. J., Trout, D., King, B., Mjolsness, E., and Wold, B. J. (2005). A mathematical and computational framework for quantitative comparison and integration of large-scale gene expression data. *Nucleic Acids Research*, 33(8):2580–2594.