A Framework for the Analysis of Gene Expression Data

Student: Don A. Frick
Tutors: Amela Prelić, Stefan Bleuler
Professor: Eckart Zitzler

Winter Semester 2003 / 04

Abstract

This paper describes the design process of a software framework for use of a biclustering algorithm for gene expression data. Facilitating the use of such an algorithm requires tools for reading expression data from files, processing said data and visualizing the results. The software must be designed in a modular fashion to allow for future expansions.

Contents

Chapter 1

Introduction

1.1 Motivation

One of the major challenges in modern molecular cell biology lies in exposing whole genomes to a set of conditions, and gaining useful information from the measured results. *Microarray technology* is a recent development that allows biologists to perform massively parallel experiments on entire DNA sequences at once. The results of these experiments are matrices that contain an expression value for every combination of genes and experiments; but due to the huge number of samples and the way genes are thought to interact with each other, it doesn't make sense to look at these values in isolation.

A popular approach to this problem is to look for groups of genes that perform similarly under a certain set of conditions. The sheer amount of data that needs to be analyzed practically requires the use of a computer as an aid. Many different clustering algorithms are in use today, but one common drawback is that they only work in one dimension at a time. Thus, they search for clusters in the set of genes and in the set of chips, but not in both at once. This artificially limits the pool of results, since there exists a large number of clusters that only span a subset of genes and chips. For example, for a particular set of genes to qualify as a cluster, it has to exhibit similar behavior accross the entire set of experiments.

Biclustering, a method recently proposed in [1], is the focus of this project. The difference to previous clustering methods is that the expression value matrix is sorted both by rows and columms at the same time. Although the goal, as always, is to find permutations that result in uniform blocks of data being displayed in the expression matrix, biclustering yields many more results since it works on rearranging genes and chips simulteneously.

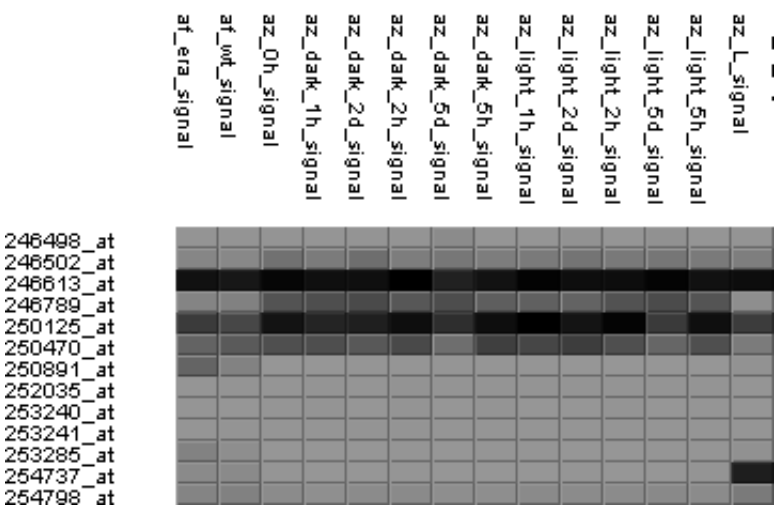

Figure 1.1: Microarray data arranged in a matrix

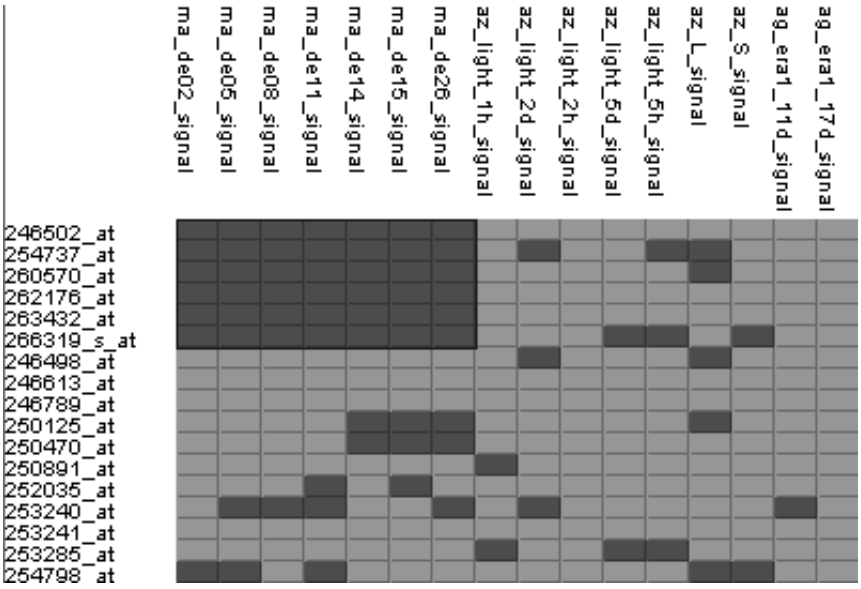

Figure 1.2: Binarized data rearranged to display a bicluster

1.2 Problem Description

The goal of biclustering is to find submatrices in the expression data that have maximal size and maximum *similarity* of values. This similarity can be measured in different ways; one of the simplest methods is to separate all expression values along a threshold, and define all values on one side to be similar to each other. A biclustering algorithm can then operate on the binarized data, and search for submatrices of values that are all above or below the threshold.

Bica is an implementation of a biclustering algorithm described in [2]. It operates on a binarized version of the expression matrix and is capable of finding all biclusters in a given data set.

Biclustering a data matrix that consists of dozens of chips and thousands of genes can result in tens of thousands of biclusters, far too many to sort through by hand. It would be useful to have a tool that assists biologists in sorting through these results and looking for those that seem to be most relevant. Due to the clustering approach, the size of the datasets and the nature of microarrays, it is also required to visualize the expression data.

The goal of this term project was to develop a framework for Bica that would facilitate use of the algorithm, and provide additional functionality to support biologists in their research.

1.3 Overview

The following chapters are presented in chronological order. *Existing Programs* will describe some currently available software packages we examined before beginning to work on the program. *Theory* will concentrate on the requirements and the planning stage of the project. The chapter on *Practice* will focus on the implementation of the code, while *Future Additions* will describe features that might be added to the program at a later date. Finally, we will present a short *Case Study* that demonstrates one of the uses of the program and end this report with a chapter on *Conclusions*.

Chapter 2

Existing programs

There exist many programs, both commercial and free, that are designed to visualize and cluster the results from microarray experiments. However, we are only aware of one other software package that implements a biclustering algorithm. The following is a short list describing some of these programs, in no particular order:

2.1 Expander

Expander is developed by Prof. Ron Shamir's Computational Genomics Laboratory at the School of Computer Science, Tel Aviv University, Israel. It implements several different clustering algorithms, can perform biclustering and provides facilities for visualization and functional analysis. Its biclustering algorithm, SAMBA, uses statistical models to limit the search to what are likely to be the most significant results. Thus, execution time is shortened at the cost of completeness. The software is freely available for academic use.

http://www.cs.tau.ac.il/~rshamir/expander/expander.html

2.2 Genesis

Genesis, developed by Alexander Sturn at the Institute for Biomedical Engineering at the Graz University of Technology, has implementations of most of the popular clustering algorithms such as K-means, self organizing maps, principal component analysis and correspondence analysis. It also has a very clean and intuitive user interface.

http://genome.tugraz.at/Software/GenesisCenter.html

2.3 GeneXPress

GeneXPress, from Eran Segal et al. at Stanford University, is a simple and clean visualization tool that provides most of the features one would desire. It is freely available for academic use.

http://genexpress.stanford.edu/tutorials/start_gxp_demo.html

2.4 TM4 suite

The TM4 suite of tools from The Institute for Genomic Research (TIGR) is unusual in that its source code is available for download. It consists of four applications, of which the Multiexperiment Viewer, a flexible Java application that handles visualization, was of the most interest to us.

http://www.tigr.org/software/tm4/mev.html

2.5 Microarray Explorer

Microarray Explorer, developed at Sourceforge, is another open source project written in Java. Its focus is on interactive data mining, with access to remote genomic databases. It offers several clustering options and a large set of visualization tools, including array pseudoimages, scatter plots, histograms and expression profile plots.

http://maexplorer.sourceforge.net/

After making some inquiries, we decided that the current program should be written from scratch, only using code from other projects if absolutely necessary.

To our knowledge, Bica is the only existing implementation of a biclustering algorithm for gene expression data that delivers a complete set of results. Writing a completely new framework would allow us to tailor it to the needs of this new algorithm, so that it can be used to its fullest potential.

This would mean more work, but would allow more flexibility in designing the code. Doing so would also avoid the pitfall of having to work through rather large codesets in order to understand them.

Chapter 3

Software Architecture

3.1 Preconditions

The core of the project, the biclustering algorithm Bica, was already complete when we started working on its graphical user interface. In its standalone version, Bica is a command line controlled Java program that reads a two dimensional array of binarized[1] data from a given file, and produces a list containing all the biclusters it has found therein. Since the current version of the algorithm operates on binary input, the data must be processed and binarized externally before one can proceed with clustering. A direct consequence of this is that the preprocessing steps can potentially have a large impact on the biclustering results.

One of these preprocessing steps is to normalize the data. Within a microarray, every chip is assigned a *control chip*, essentially a set of values that it should be normalized to before being processed any further.

3.2 File formats

The unprocessed microarray data is read from files, the exact format of which was to be determined during the course of the project. Although there is no clear standard, most existing data seems to conform to the following rules: The data is contained in plain text ASCII files, with the first line containing the chip names. One of the first words of each following lines is the gene name, followed by the expression values for that gene. All names and values are separated by one whitespace, either a space character or a tab character. Two consecutive whitespaces signify a missing value.

A second file relates each chip to its control chip. Each line in this file contains the name of a chip, followed by the name of its control chip, separated by whitespaces.

[1] If the value is above a given threshold, it becomes a one; else, a zero.

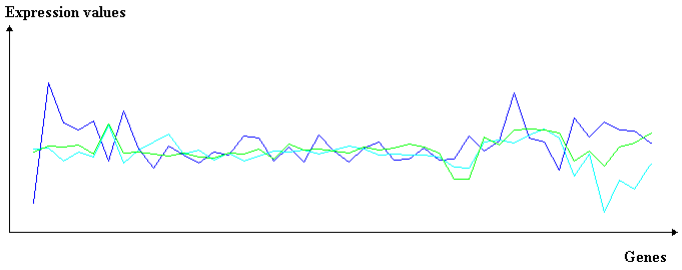

Figure 3.1: Expression graphs

The files may contain additional information, such as gene pathways, that was not used in the course of this project.

3.3 General design goals

The fundamental goal of providing a framework for Bica and other clustering algorithms can be divided into two parts: A graphical user interface, and the underlying data processing code.

3.3.1 GUI requirements

Perhaps the most important part of the GUI is the visualization of the data. It should include a display of the data matrix, with the expression values mapped to a color scale[2]. The visualized matrix should have name tags for the gene rows and chip columns, a way to mark genes and chips for later reference, and functionality for zooming in and out of the display. The program needs to be able to rearrange the matrix so that a bicluster is clearly displayed to the user. It should also be possible to retrieve detailed information on biclusters and particular gene-chip combinations within the matrix.

The user also should be able to view expression graphs for each gene, or a selected set of genes. These graphs plot the expression values of a particular gene accross the various experiments; they are generally not interesting for the absolute values that they plot, but because they allow a comparison of the behavior of several genes accross the entire set of conditions.

3.3.2 Data processing framework

Before commencing any work, the program needs to retrieve all necessary data from properly formatted files. The expression data sets are then normalized to their control sets. This is done by dividing each expression value of a chip by

[2] e.g. low values are red, high values are green

the corresponding value in its control chip. The control chip values are then removed from the dataset. If the user wishes, the data can then be logarithmized to a base of two or ten. Another optional step would be to perform some form of statistical filtering on the data. Finally, the data must be discretized using a user inputted threshold, which results in a matrix of binary values. Missing values are marked, both for the biclustering algorithm and for later reference.

The next step is to run the biclustering algorithm on the processed data, which results in a list of biclusters. At this point, it would be interesting to look for extensions to the biclusters that have been found; these are uniform blocks that can be appended to a bicluster without necessarily completing a larger one.

In order to help the user find the most interesting results, the program needs scoring schemes that allow a ranking of the biclusters. The user should be able to sort the entire list of biclusters according to the various ranking schemes, and search for biclusters that fulfill certain criteria. There should also be an option to search for genes by their names, and to get a list of the biclusters that contain these genes. In a similar vein, it would be interesting to rank genes according to their participation in various biclusters, and perhaps even associate genes with each other by looking at how many clusters they share.

3.4 Basic design decisions

The programming langauge chosen for the project is Java. It offers platform independence, a large toolbox that includes support for graphical user interfaces, and a programming environment that I was already familiar with. The core biclustering algorithm is also written in Java, with several supporting Perl scripts that perform preprocessing and formatting steps. We decided to implement the entire program in Java rather than reuse those scripts.

We originally planned to separate the program into a set of five functional parts, each of which would perform a set of tasks that are closely associated with each other. The list of parts included:

- A file reading object, responsible for reading and writing to the hard drive.
- One central manager for the raw data matrices, where the data would be kept and preprocessed (normalization, discretization etc.)
- An instance of Bica, to perform the biclustering.
- A postprocessing unit where the list of results (biclusters) would be kept for sorting and searching.
- A visualization unit, responsible for displaying the data matrix, the expression views etc.

Doing so would give the code some structure and modality, making it easier to understand and modify. Having the central manager and the GUI control the other parts of the program also seemed like a good idea, since having one

governing body should significantly reduce the number of errors made in writing the code.

In terms of the GUI, we originally envisioned one large window containing several smaller, resizeable windows that would display various lists and display modes. But since this version doesn't offer any appreciable advantages over a simple split pane, we quickly abandoned it. The layout we settled on in the end is as follows:

- A main menu bar at the top of the window, which allows the user to change settings and invoke commands
- A left hand pane, which contains a list of various display modes that the user may select from
- A right hand pane, which displays whatever the user currently has selected (currently either a data matrix or a set of expression graphs)

This setup is both simple and functional. It isn't very flexible in that it only allows the display of one graphical representation at a time, but the tradeoff is that all the data is presented in only three distinct areas, which should make for a very simple and user friendly experience. It is also easily extensible; the left hand pane can include any number of display options, and the right hand pane can be used to draw any customized graphics that are required.

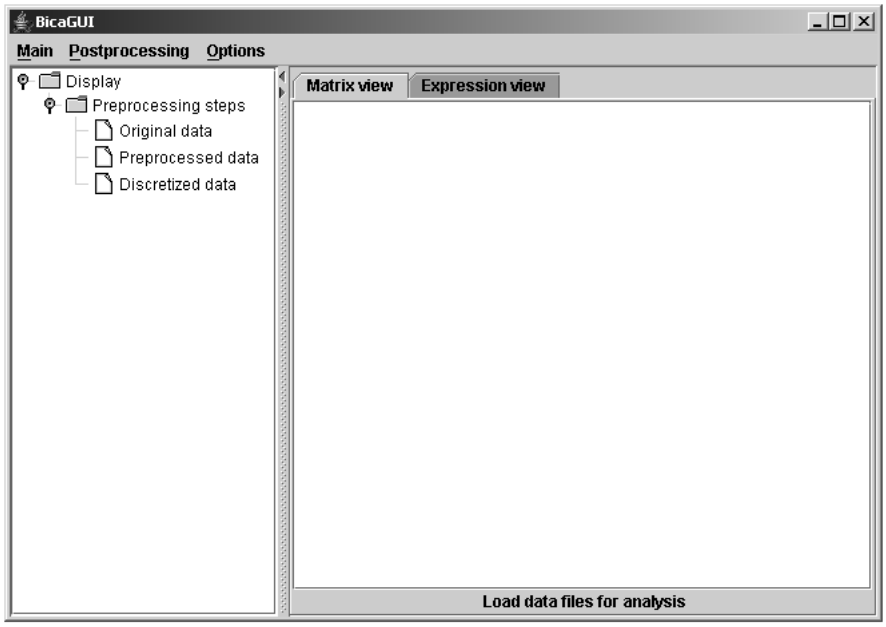

Figure 3.2: The GUI

Chapter 4

Implementation

4.1 Preprocessor

The preprocessor, as it stands, is a combination of the file reading and preprocessing units we had originally planned to make. Since the steps of reading the data and performing initial operations on it are closely related and often done right after each other, they were included in a single object.

The file reading code was fairly straightforward to implement; one of the problems encountered while writing it was how to determine the size of the input file efficiently. A first solution involved dynamically resizeable arrays, but we later included a second version that required the file size to be included in the header. Since the files can contain tens of megabytes of data, it seemed sensible to make such a requirement for the sake of speed. Other than that, the format required for the input files is identical to what was described in section 3.2. The result of the file reading steps is a two dimensional array that contains the expression values, and several other arrays containing gene and chip names and control chip relations. It should be noted that this raw data is kept until a new batch of files is loaded; this allows the user to repeat the preprocessing steps an arbitrary number of times, and even view biclustering results with different preprocessing options.

The preprocessing steps themselves are quite simple, often just requiring one operation to be performed on every element of the data array. Each step is implemented as a separate function, which should make it easy to make modifications or add new steps. One small pitfall that one should keep in mind is that the original data contains a number of chips that are control values. These are the values that the actual experiment results are normalized to, and as such, they are discarded after use. Skipping the normalization step simply removes these chips from the matrix, but either way, the matrix ends up with less columns than it had before. This has some implications later on, especially when it comes to visualizing the data.

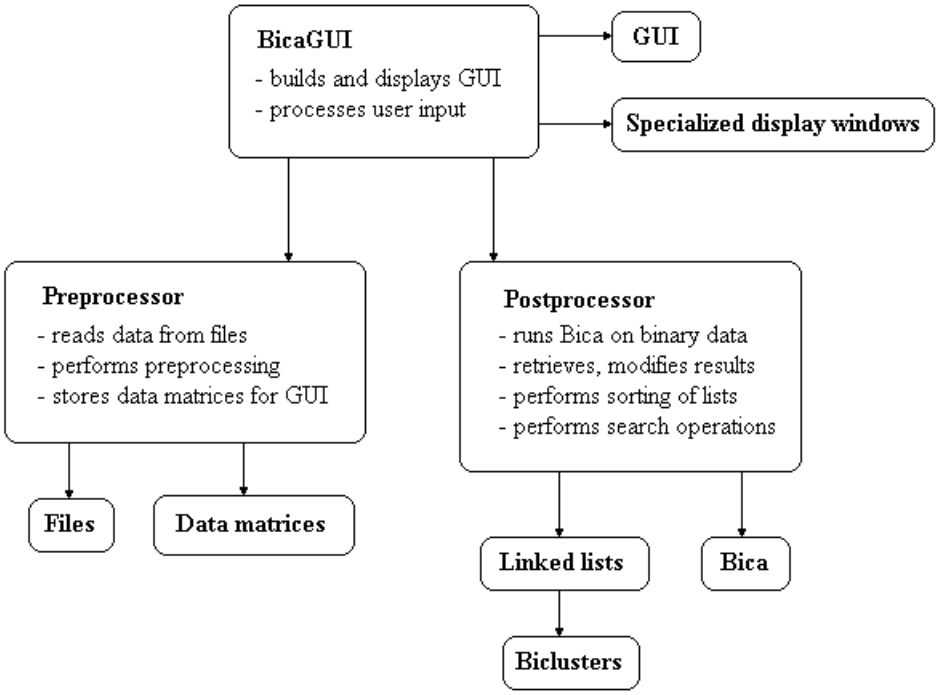

Figure 4.1: Class structure of the program

4.2 Postprocessor

The postprocessing unit is responsible for running the biclustering algorithm, or, for that matter, any other algorithm on a preprocessed data set. It then stores the list of resulting biclusters and performs a searching and sorting commands on said list.

Since running Bica is a very processing intesive task, it is executed in a thread that is separate from the GUI and the other processing units.

The central piece of this class is a linked list of Bicluster objects. Each of these objects contains complete information on one bicluster, including the genes and chips involved, the total size and the Mean Square Residual Score. The program will keep a copy of the original list at all times, so the user can perform search operations on the full list as often as needed. The methods that operate on the list will simply traverse it, either looking for biclusters that meet certain restrictions or sorting the list according to some given criteria.

The Mean Square Residual Score (MSRS) of a bicluster is one of the criteria used to sort the list of results. The following definition is taken from [1]:

Let X be the set of genes and Y the set of conditions. Let a_{ij} be the element of the expression matrix A representing the logarithm of the relative abundance of the mRNA of the ith gene under the jth condition. Let $I \subset X$ and $J \subset Y$ be subsets of genes and conditions. The pair (I, J) specifies a submatrix A_{IJ} with the following *mean squared residue* score.

$$H(I,J) = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2,$$

where

$$a_{iJ} = \frac{1}{|J|} \sum_{j \in J} a_{ij}, \quad a_{Ij} = \frac{1}{|I|} \sum_{i \in I} a_{ij},$$

and

$$a_{IJ} = \frac{1}{|I||J|} \sum_{i \in I, j \in J} a_{ij} = \frac{1}{|I|} \sum_{i \in I} a_{iJ} = \frac{1}{|J|} \sum_{j \in J} a_{Ij}$$

are the row and column means and the mean in the submatrix (I, J). A submatrix A_{IJ} is called a δ *-bicluster* if $H(I, J) \leq \delta$ for some $\delta \geq 0$.

Thus, a high MSRS indicates that the values within the bicluster vary over a large range of values, while a bicluster with very uniform values will have an MSRS close to zero.

One thing to keep in mind is that the Bicluster objects gained from the biclustering algorithm Bica are not the same as the Bicluster objects used by Postprocessor. This is an unfortunate naming problem, but shouldn't cause any problems if kept in mind. As it stands, the list of results has to be transformed into a format useable by Postprocessor before the program can perform searching and sorting operations.

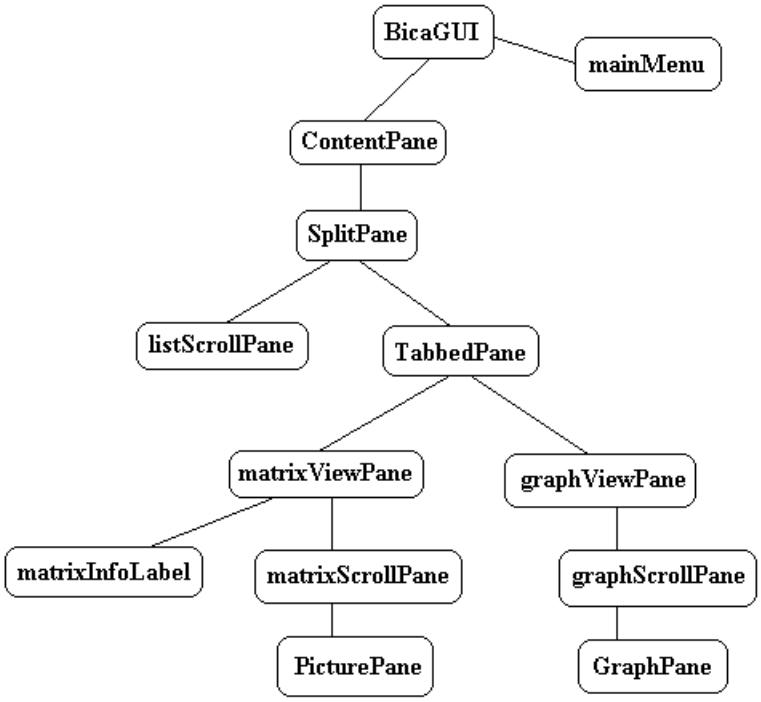

Figure 4.2: Code structure of the GUI

4.3 BicaGUI

This central unit builds the graphical user interface, processes user input and controls both the Preprocessor and the Postprocessor.

The GUI itself is a fairly striaghtforward implementation of what Java's Swing toolkit offers. The only unusual part is to the right hand of the split pane, where one of two customized graphics panels are displayed.

The first of these is responsible for drawing the visualization of the data matrix. It also keeps track of user selections within the matrix so that the corresponding rows and columns may be marked. If the user selects a bicluster for display, the graphics panel will build two translation tables, for genes and chips respectively. These tables are simply arrays that contain a rearrangement of the gene and chip indices, thus describing a permutation of the matrix that will bring the bicluster out at the top left corner. Gene and chip names are drawn along the top and left hand borders, so all matrix coordinates have to include an offset to keep the elements from overlapping. The matrix itself consists of a

block of small rectangles, each of which has the color shade corresponding to the expression value it represents. The zooming options simply make the program redraw the contents of this panel at a different size.

The second display panel shows the expression graphs of genes that the user has selected within the matrix. The scaling of these graphs was kept simple and rather arbitrary since most users will only need qualitative information. The graphs for the selected genes receive one out of a set of predefined colors, the same one as in the selection in the data matrix. However, this set of colors is limited, so the selection of a large numbers of genes will cause a counter to wrap around and start repeating. Another option would have been to map the selections to a color scale (e.g. green - blue), but it turned out that this approach results in many similar shades, which is perhaps even more confusing.

One of the issues we encountered was that redrawing large data matrices can put a large strain on the system. A simple fix, included in the program, is to limit the number of gene rows that are displayed. Since the point of interest will generally be a bicluster which is displayed toward the top left corner, it's an obvious sacrifice to make.

The left hand panel contains the list of display options that are currently available to the user. These options are built into a tree, which allows for almost limitless extension of the list and easy grouping of elements. The original plan was to generate a table of all biclusters that the user would be able to sort according to different criteria by clicking on the column headers. However, this approach turned out to be too difficult to implement, and too inflexible in regards to future extensions. As it is, the display tree can hold an original master list of results and a separate list of search results, not to mention entries for any additions that might be made later on.

One question that had to be asked early on was how to structure the workflow of the program, that is, how the user would progress through the different processing steps that are provided. We had hoped to make this as simple as possible by consolidating the file loading and preprocessing steps, allowing the user to get to the results by executing just two commands. This turned out to be a bad idea since biclustering is such a time consuming process. In the end, we decided that it would be better to let the user try out different preprocessing options and view the results before handing them to the biclustering algorithm. An added benefit is that the user can view a set biclustering results under different preprocessing conditions without having to execute the Bica algorithm every time.

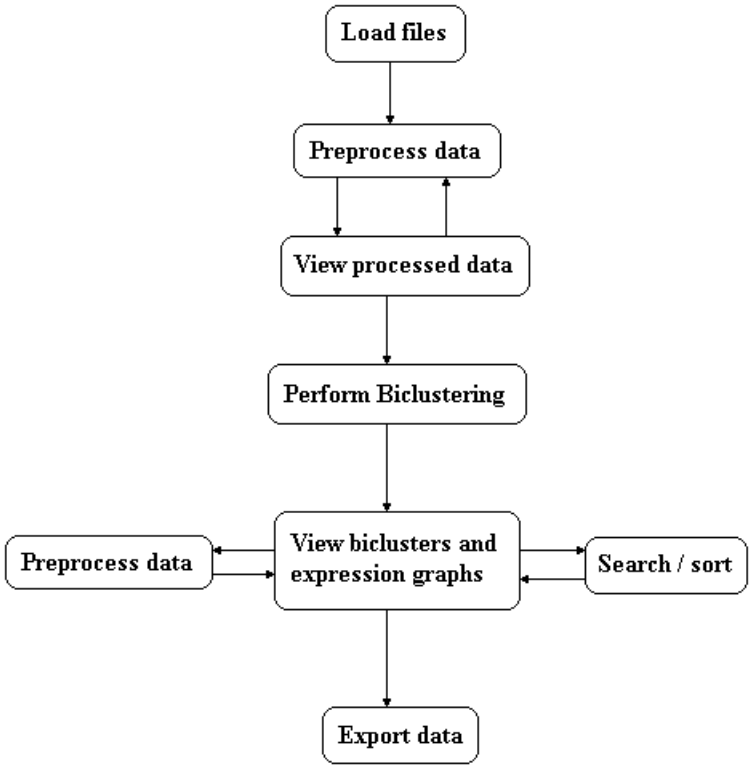

Figure 4.3: Workflow of the program

Chapter 5

Future additions

Although the program is completed and useable at its current stage, there is a large number of items that could be added to increase its functionality.

One obvious thing to do would be to add more clustering algorithms. Although biclustering is a powerful tool, it would be beneficial to compare its results to those of other algorithms, and perhaps see how much they overlap. In a similar vein, it might be interesting to repeatedly perform biclustering on a dataset with different preprocessing options activated each time, and see how the results relate to each other.

The scoring algorithms are another area that could easily be added to. While the currently implemented scoring schemes should cover basic needs, there are many others that would likely be worth adding. One of the more interesting ideas is to assign a *suspiciousness score* to biclusters. This is essentially a measure of the likelihood that a bicluster is a purely random assortment of genes and chips that just happen to have similar values. If the suspiciousness is high, then that uniform collection of samples in the matrix is more likely to be significant in some way or another.

Some of the things listed under desired features also remain to be added. Finding extensions to biclusters would certainly be interesting, since the current algorithm, for all its beauty, is a bit restrictive; since it works with strictly binarized data, it will fail to recognize a large bicluster that contains just one value that is on the wrong side of the threshold.

On a related note, one could add functionality to display several biclusters simultaneously. This would require the program to check whether a given set of biclusters can be shown without any conflicts occurring.

Chapter 6

Summary and Conclusion

We have designed and implemented a framework for a biclustering algorithm for gene expression data. The two goals of the project were to design an interface to facilitate use of this algorithm, and to make a framework that would allow the addition of other data mining and processing tools.

We believe that the goal of usability as been attained, as the program has reached a stage of completion where most of the necessary features have been implemented. The graphical user interface is capable of displaying gene expression data sets in the most widely used formats, with the additional functionality required to work with large numbers of biclusters.

The underlying code implements a set of essential tools for processing and navigating given datasets and clustering results. The software was written and documented with future expansions in mind, and as such should be well suited to accomodate any future additions.

The distinguishing feature of the biclustering algorithm we used is that it is capable of finding all the maximal biclusters in an expression matrix. It is, to our knowledge, the only available program that offers this functionality.

List of Figures

Bibliography

[1] Yizong Cheng and George M. Church. Biclustering of expression data. pages 93–103, 2000.

[2] Amela Prelić, Stefan Bleuler, Philip Zimmermann, Peter von Rohr, Anja Wille, Peter Bühlmann, Wilhelm Gruissem, Lothar Thiele, and Eckart Zitzler. Studying the Usefulness of Biclustering on the Global Scale. *Submitted for publication*, 2004.