Fall 1-31-2003

Functional annotation and dendrogram representation of gene expression clustering results

Antoneta Petkova Vladimirova
New Jersey Institute of Technology

ABSTRACT

FUNCTIONAL ANNOTATION AND DENDROGRAM REPRESENTATION OF GENE EXPRESSION CLUSTERING RESULTS

by
Antoaneta Vladimirova

The advances in genomic sciences have created vast amounts of gene expression data. To make sense of the expression information, various techniques have been applied. Clustering is among the unsupervised methods used to group the results according to gene expression level. Dendrogram visualization allows graphical representation of the clustering. The aim of this thesis is to enhance these techniques by adding another layer of functionality, namely, annotating the dendrogram with gene functional information. Presented is an application which visualizes yeast clustering results as a dendrogram along with color-coded gene keyword annotations. Gene keyword information was extracted from a major biological database and was used to create a database which was queried by the program according to the user preferences. Functional annotation with keyword information will help the biologists to integrate the different type of visual information quickly and provide an intuitive way of correlating the gene expression results with gene function.

FUNCTIONAL ANNOTATION AND DENDROGRAM REPRESENTATION OF GENE EXPRESSION CLUSTERING RESULTS

by
Antoaneta Petkova Vladimirova

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer Science

January 2003

APPROVAL PAGE

FUNCTIONAL ANNOTATION AND DENDROGRAM REPRESENTATION OF GENE EXPRESSION CLUSTERING RESULTS

Antoaneta Petkova Vladimirova

Dr. Michael Recce, Thesis Advisor ' / Date
Associate Professor, Department of Biomedical Engineering, NJIT

Dr. Barry Cohen, Committee Member Date
Assistant Professor, Department of Computer Science, NJIT

Dr. Peter Tolias, Committee Member Date
Associate Professor, Department of Microbiology and Molecular Genetics
UNDNJ-New Jersey Medical School

BIOGRAPHICAL SKETCH

Author:	Antoaneta Petkova Vladimirova
Degree:	Master of Science in Computer Science
Date:	January 2003

Undergraduate and Graduate Education:

- Master of Science in Computer Science
  New Jersey Institute of Technology, Newark, NJ, 2003
- Doctor of Philosophy in Cell Biology
  New York University, NY, 2000
- Master of Science in Cell Biology
  New York University, NY, 1995
- Master of Science in Genetic Engineering
  Sofia University, Sofia, Bulgaria, 1992

Major: Computer Science

Publications:

Vladimirova A.,
"Mechanisms of E47 transcriptional regulation,"
Ph.D. Dissertation, NYU, September 2000.

Prabhu S, Ignatova A., Park S.T. and Sun X.-H.,
"Regulation of the expression of cyclin-dependent kinase inhibitor p21 by E2A and Id proteins", MCB 17(10), pp. 5888-5896, October 1997.

Ignatova A., Xu M., Sun X.-H.,
"The role of E2A transcription factor as a master regulator of B-cell development, "The FASEB Journal, 10(6), pp. 923-923, April 1996.

Ignatova A.,
"Studies on genetic transformation of tobacco via *Agrobacterium tumefaciens* with vector carrying the gene of capsid protein of PPV (Plum Pox Virus),"
M.S. Thesis, Sofia University, May 1992.

Note: Family name Vladimirova corresponds to maiden name Ignatova.

ACKNOWLEDGMENT

I would like to thank Dr. Michael Recce for being my thesis advisor and his suggestions. I am very thankful to Dr. Barry Cohen and Dr. Peter Tolias for their participation in my review committee and their valuable recommendations. I really appreciate the support and advice that Georgy Vladimirov, Diego Sandoval and Vanio Ivanov have given me when I really needed it.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

LIST OF ABBREVIATIONS

EBI	European Bioinformatics Institute
ER	Entity-Relationship
CDS	Coding Sequence
DNA	Deoxyribonucleic Acid
GEO	Gene Expression Omnibus
GO	Gene Ontology
GUI	Graphical User Interface
JDBC	Java Database Connectivity
KEGG	Kyoto Encyclopedia of Genes and Genomes
mRNA	messenger Ribonucleic Acid
ORF	Open Reading Frame
PCA	Principal Component Analysis
SIB	Swiss Institute for Bioinformatics
SGD	Saccharomyces Genome Database
SMD	Stanford Microarray Database
SOM	Self-Organizing Maps
SQL	Structured Query Language
UPGMA	Un-weighted Pair Group Method Using Arithmetic Averages

CHAPTER 1

INTRODUCTION

1.1 .Objective

The objective of this thesis is to build a tool to assist in the interpretation of the microarray gene expression experiments. It aims to add more functionality to visualization of gene expression clustering results by graphically annotating the genes. Clustering techniques and dendrogram visualization are widely used to analyze gene expression data. In this work another layer of functionality was added to the expression analysis tools by providing color-coded functional annotations of the genes in the dendrogram.

With the advance of modern genomic sciences, a wealth of gene expression information is generated which needs careful analysis to extract knowledge from the raw biological data. Gene expression of thousands of genes is representative of the state of the cells, cellular processes and gene product inter-relations. However, human beings cannot process effectively numeric information in such quantities. Various analysis and visualization tools are needed to simplify the process of data interpretation. Gene clustering methods have been widely applied to effectively group genes according to their expression level. The assumption being made is that genes with similar expression level that are grouped together could be involved in the same cellular processes, could be co-regulated or other functional relationships could exist among them. To facilitate the researcher looking at the clustered data in a form of a dendrogram, an application was designed which enhances the visualization of the clustering results with functional biological information. Along with the dendrogram keyword gene annotations are displayed in a color-coded scheme according to the user preferences. This can provide the biologist with fast and intuitive means of quickly evaluating the functional information. It can save the researcher a long and tedious process of manually looking up the entries of each gene in which he is interested in and can serve as a first line of reference and selection of interesting genes to focus on.

1.2 Background Information

Presented is a bioinformatics tool for analysis of gene expression data. To understand the specific approach taken, some background information concerning the microarray technique, clustering and visualization methods and gene annotation tools will be discussed. Biological and data mining concepts and techniques are presented to set the conceptual frame in which the application tool is developed.

1.2.1 Gene Expression and Microarrays

With the advent of DNA microarray technologies, large-scale gene expression experiments have been widely performed in the recent years. The development of the microarray techniques allows the researchers to obtain vast amounts of gene expression data, which needs to be stored, maintained and analyzed automatically. Hundreds or thousands of genes are studied simultaneously, providing information about gene expression patterns in cells in various physiological conditions such as normal versus diseased state. Cells representing different stages of development as well as cells treated with drugs or maintained in different media could yield enormous amount of information which has to be mined to extract deeper knowledge about the undergoing processes and molecules that control them. Understanding patterns of expressed genes will improve our comprehension of the complex networks in the biological systems (Brazma and Vilo, 2000).

One of the most popular platforms allows the comparison of the abundance of mRNA in two samples, one of which is used as a control. Briefly, RNA from the sample and control is extracted and labeled with different fluorescent labels. The cellular extracts are placed over the microarray plate which contains complementary DNA fragments and are allowed to competitively hybridize. When excited by a laser, the fluorescence intensities reflect the relative expression level of the genes in the sample and control. The ratio of the fluorescent intensities read is used for further analysis.

Another platform utilized by Affymetrix measures the mRNA level from two samples separately. Each sample's mRNA labeled molecules are placed on separate microarray plates, allowed to hybridize and their signals read and then compared. Again, the ration between the two signals is used for further studies.

The gene expression data from various experimental conditions may be represented as a so-called "gene expression matrix" (Brazma and Vilo, 2000). The matrix can be imagined as a table. In the table the rows represent different genes and the columns correspond to the conditions under which the cell was studied, e.g. from various developmental stages, from different tissues or treated in a particular manner. Each cell in this table contains a number that characterizes the expression level of the gene corresponding to the given condition. Some universities, institutions or governmental organizations currently maintain huge databases of such tables for use by the scientific community. To transform the raw images into table entries, a variety of techniques is employed to identify the spots corresponding to genes on the microarray, to determine their boundaries, to measure the fluorescence intensities from each spot and the background. The pre-processing of the data is a non-trivial task and the outcome of the analysis may depend to a great extend on the initial preparation of data (Quackenbush, 2001). The expression matrix is further used for clustering and can be visualized for better understanding of the values in a color-coded scheme. Traditionally, the up-regulated gene expression is denoted in red, the down-regulated expression – in green, and gene expression with no changes – in black.

A very important fact should be emphasized when studying gene expression. A researcher is ultimately interested in the function and interactions of proteins which carry out the gene function. Gene expression experiments, although much easier and cheaper to perform, might not be completely representative of their products (Brazma and Vilo, 2000). The rate of mRNA synthesis might not reflect the protein amount. In addition, post-translational modifications of the proteins, which are independent of transcription level, are extremely important for the function of the gene product. It should also be noted, that the microarray-generated data is still not quantitative in terms of mRNA molecules, but reflects relative expression of genes. Thus, the outcome of applying quantitative methods might not be completely accurate.

Since these experiments reflect the international efforts to decipher gene expression implications, there is an urgent need for standardization of experimental settings as well as annotation and submission procedures. MIAME or Minimal

Information about a Microarray Experiment is recently described by Brazma and colleagues (Brazma et al., 2001).

Gene expression data should not be studied independently, but viewed in the light of the rest of experimental data available. In order to make valid conclusions, gene expression analysis must be combined with other existing biological information obtained by other independent methods. In the presented application, keywords information extracted from a major biological database was used to complement the results obtained by clustering the expression data.

1.2.2 Clustering Methods

Clustering of the data according to some measure of similarity is one of the data mining techniques currently used for analysis of microarray expression data (Celis et al., 2000). The output is clusters of genes with similar patterns of expression. The grouping of genes with known and unknown functions according to their expression level might indicate a functional role for the unknown genes thus potentially leading to targets for further biological research. The underlying assumption made is that genes that are co-regulated show similar patterns of expression and participate in the same pathway or respond to the same environmental signals. Clustering may lead to better understanding of gene regulation in norm and disease, to deciphering of metabolic and signaling pathways, to reverse-engineering of gene networks and better drug treatments design.

There are different approaches of clustering the gene expression data, unsupervised and supervised. Another categorization divides them into agglomerative and divisive depending on whether the algorithm starts with individual members and fuses them together in one big cluster, or, respectively, starts with all data and gradually divides it in groups.

In the unsupervised methods, just the gene expression levels are taken into account when performing the clustering. In the supervised techniques, some previous biological knowledge is also incorporated, e.g. which genes should cluster together (Brazma et al., 2000 and Quackenbush, 2001). The supervised methods act as classifiers of a data set based typically on positive and negative training sets to teach the classifier initially. For example, Califano et al. designed a classifier to predict cell phenotype and drug sensitivity. Support Vector Machines (SVM) is another approach (Brown et al., 2000) to predict functional roles for uncharacterized yeast open reading frames (ORFs). The goal is to construct classifiers which assign a given expression profile to a predefined class. This can be very useful for diagnostics when classifying tumor versus normal samples and in patient prognosis determination.

Many different unsupervised clustering algorithms exist such as hierarchical clustering, agglomerative clustering, K-means clustering, self-organizing maps, principal component analysis and others, but there is no consensus as to which one is the best. It seems that different algorithms are better- or worse-suited depending on the type of data analyzed.

The unsupervised clustering algorithms, in particular the hierarchical clustering methods are of most interest for this project. However, a brief introduction to some other unsupervised and supervised methods will be given for completeness.

The data points are typically represented by the $\log_2$ (ratio), where ratio is the normalized value of the expression level for a particular gene in the query sample divided by its normalized value for the control sample (Quackenbush, 2001). Each gene is represented as an expression vector and the dimensionality of the expression space is specified as the number of separate genes per experiment. Genes with similar expressions throughout the same set of experiments will be positioned close in the expression space. Alternatively, each data vector might represent the values of the different genes during one experimental condition and the dimensionality in this case is modeled by the number of genes examined. In this case, gene expression profiles similar for different experimental conditions will be clustered together.

The idea of the K-means algorithm is as follows: initially all the data is arbitrarily divided into *n* clusters, where *n* is specified a priori. Then the center of each cluster is computed and the distances between the data points to the cluster centers are calculated. The data points are assigned to the closest cluster according to a minimum distance to the cluster center. Distance metric, usually Euclidean distance or correlation coefficient, is used for calculation of distances between the points. Then the cluster centers are re-calculated. The process is repeated until no further cluster re-assignments are made or until a predefined number of iterations have been reached.

As an alternative or in addition to the K-means clustering, a hierarchical grouping can be performed. Hierarchical clustering or UPGMA (un-weighted pair group method using arithmetic averages) joins single gene expression profiles to form a hierarchical tree based on a similarity measure. First, the pair-wise distance matrix is calculated for all of the genes in the same cluster. Second, the distance matrix is searched for the two most similar clusters (initially a cluster consists of just one gene). Third, two selected clusters are merged into a new cluster. The procedure is repeated iteratively until only a single cluster is left. The joined genes can be represented as leaves of a hierarchical tree (dendrogram), and after the join a new node is created thus assembling the tree from the leaves towards the root. The heights of edges of the tree reflect the distance between the genes that formed that node. This technique produces a hierarchical tree to visualize the results (Jagota, 2001).

Self-organizing maps (SOM) is a neural network-based divisive clustering method and Principal Component Analysis (PCA) is a mathematical technique that allows visual representation of data and effectively reduces the dimensionality of gene-expression space without significant loss of information (Quackenbush, 2001).

In any clustering method, the calculation of the distance between the gene or experiment vectors is one of the most important criteria according to which different genes or experiments are grouped together. Different distance measures, or distance metrics may produce dissimilar clustering (Quackenbush, 2001). In the single-linkage clustering, the distance between two clusters is the minimum distance between two members of two clusters. This technique is, therefore, referred to as nearest-neighbor method. Among the other techniques is the average linkage clustering, or UPGMA, where the average distance is calculated from the distance between each point in a cluster and all other points in another cluster (Durbin et al., 1998). Measuring similarity or dissimilarity is reviewed in Jagota, 2001.

Unsupervised methods are the most widely used clustering methods. They do not take into account any additional biological information into account when computing the similarities among genes. The researchers are usually doing further research on the interesting features of the clustering output manually, which is very time-consuming and tedious. This may involve going through multiple databases maintained in different formats and containing different data. One can be easily lost in the process. Automation of the process of annotation of the clustered data is necessary to highlight the biological characteristics of the studied genes. Information extracted from the existing biological databases and applied to the clustering method can augment the knowledge obtained from the analysis of the expression profiles.

Clustering methods for analysis of microarray data are implemented and available for download or are web-accessible. Michael Eisen's Cluster software implements different clustering algorithms (Eisen et al., 1998). The software is freely available and can be downloaded at http://rana.lbl.gov/EisenSoftware.htm. For a set of genes, an upper diagonal similarity matrix is computed according to a user-specified metric. The matrix contains similarity scores for all pairs of genes. The matrix is scanned to find the highest value reflecting the most similar genes and a parent node is created that joins them. This procedure is repeated until only one node remains. The output of the program is a .cdt file with the clustering information calculated. The file can be used by TreeView software (Eisen et al., 1998) to visualize the clustering results. FreeView, a Java implementation of TreeView can be found at http://magix.fri.uni-lj.si/freeview/ and was developed by Marko Kavcic and Blaz Zupan. The source code is freely available and was used and modified for the purposes of the presented project to accommodate the annotation functionality. Our program adds value by supplying graphical keyword annotation information for the clustered genes to facilitate the researcher.

1.2.3 Visualization Techniques

The microarray-based technology and other high-throughput techniques produce vast amounts of numeric data which are not easy assimilated by humans. More natural and intuitive methods are needed so that the complexity of the information is preserved but it is easy for the investigator to interpret the important features.

As an initial step, Eisen et al., 1998, present the raw microarray data graphically by coloring each cell on the basis of the fluorescence measured. Cells with logarithm ratios of zero are colored black, the positive and negative ratios are colored red and green, respectively (Eisen at al., 1998). This approach has been adopted and used widely nowadays.

The visualization of the clustering results is an important step of the process since it allows researchers to grasp features or patterns that are difficult to impossible to comprehend just by looking at the data in that is in the form of numbers or even colors. The hierarchical tree/dendrogram representation of the results allows visualization of the gene relationships based on the similarity in their expression. The length of the branches corresponds to the level of similarity between the different genes.

Scatter plots and principal component analysis are other visualization techniques reviewed in Jagota, 2001.

This thesis attempts to add another level of graphical representation that is easy to grasp. The keywords functional information is color-coded according to the user preferences and displayed in the form of colored squares so that the investigator can simultaneously view the colored matrix of gene expression, the cluster and the annotation information.

1.2.4 Gene Annotation

The exponential growth of gene expression data in the recent years creates a need to sift through the sea of information to search for biological meaning. Therefore, simply applying computational tools to group of the expression patterns does not answer the question of how to interpret, make use of this information or address the complexity of biological processes. Although the assumption made is that genes with similar expression patterns share similar functions, the clustering methods do not automatically provide us with the knowledge of functional organization of genes, gene networks, co-regulation or structure of biochemical pathways.

On the other hand, supervised methods, which take into account some existing biological knowledge to classify genes with unknown functions, may lead to some misleading predictions. As it was noted above, mRNA levels may not always be representative of the level of protein which carries out the function. It does not reflect its the post-transcriptional modifications. In addition, supervised methods classify data according to some criteria chosen for the analysis and do not reveal other functional dependencies that might be interesting for the researcher. Therefore, they are very useful, for example, for classifying samples for diagnostic purposes, but are limiting in a way that it does not allow discovery of novel patterns among all the genes.

Therefore, a suitable approach would combine the two strategies of unsupervised and supervised methods in such a way that it would not be biased just as the unsupervised methods, but it will incorporate pre-existing biological knowledge to facilitate the interpretation of the results. This could be achieved by presenting at the same time clustering results along with gene annotation information and let the investigator with his expert knowledge and insight to decide how to use the available information from clustering and biological gene characteristics.

Presenting the biological properties of the gene could be done in many ways. For this project, keyword information from SWISS-PROT biological database was chosen (http://us.expasy.org/sprot/). This publicly available database has a reputation of being the most precise and accurately curated and it maintains information related to gene function such as keyword and function description. It has a vocabulary of 333 keywords related to yeast genes (Appendix B.4). Although it is impossible to incorporate the information of all the keywords along with the clustering visualization and maintain the clarity and simplicity, a small number of keywords selected by the user can be presented easily to provide helpful information and insight to the investigator. For easier visualization, the keywords can be color-coded and represented as symbols such as squares or other icons, to make it easy for the user to grasp the information.

A very systematic and large-scale method of annotating genes in different organisms is under way. The Gene Ontology project (GO) and Gene Ontology Consortium was formed to develop shared, structured vocabularies to annotate molecular characteristics of genes across organisms (http://www.geneontology.org/#ontologies). The Consortium seeks to provide a set of controlled vocabulary for specific biological domains that can be used to describe gene products in any organism. The work includes building three extensive molecular ontologies to describe molecular function, biological process and cellular component. Gene ontology may be seen as a tree, where parent nodes give a more general description of a gene than their children (Hvisdten et al., 2001). The leaf nodes give an accurate description of each gene. For each gene in a microarray experiment one or more nodes in the ontology tree may be found to represent the existing knowledge about its functions. Functional groups or classes can be retrieved from the gene ontology by traversing the tree bottom-up, starting from the leaf nodes and stopping when reaching nodes that within their sub-trees contain a group of genes labeled with the same annotation. The consortium was initiated by scientists researching yeast, the fruit fly and the mouse, but additional model organism groups are joining the project. Members of the Consortium can contribute to the updates and revisions of GO. The use of ontology methods to organize the available biological knowledge is an area of active research. The ontologies are structured vocabularies in the form of directed acyclic graphs that represent a network in which each term may be a "child" of one or more than one "parent". Relationships of child to parent can be of the "is a" or the "part of" type to represent an instance of the parent or component of the parent, respectively. Definitions of the GO terms are obtained partially from other databases such as SWISS-PROT. Keywords used in this application have been incorporated as GO terms. The mapping between the keywords and the GO terms can be found at http://www.geneontology.org/external2go/spkw2go. When GO becomes available for more organisms and becomes widely used it will be a much better alternative than keywords to incorporate into an application as the one described. It will give a much broader and comprehensive perspective that the currently used keywords.

1.2.5 Examples of Available Resources

There have been many attempts for integration of different biological analysis tools and/or information from multiple databases. One such application is the Database Referencing of Array Genes Online (DRAGON) (http://pc190-

10.kennedykrieger.org/dragon.htm). DRAGON is a web-accessible database that aids in the analysis of differential gene expression data as a biological annotation tool. This web-accessible database currently extracts or is in the process of collecting information related to the biological characteristics of individual genes from a variety of publicly available sources such as SWISS-PROT, Pfam, UniGene, Kyoto Encyclopedia of Genes and Genomes (KEGG), Online Mendelian Inheritance in Man (OMIM), Transfac, Interpro, Biochemical Image Database (BBID) and multiple yeast databases. The user submits his list of genes with or without gene expression data and request that list to be annotated with specific type of information relevant to the experimental system being studied. It returns the same text file that was submitted with additional columns containing gene annotation information. View tools are being developed to integrate the expression data and the type of annotation information requested by the user.

Other web-based services related to microarray analysis include ArrayExpress (http://www.ebi.ac.uk/microarray/ArrayExpress/arrayexpress.html), Gene Expression Omnibus (GEO) (http://www3.oup.co.uk/nar/database/summary/319), ArrayDB (http://www.niehs.nih.gov/Connections/2000/oct-nov/nov00-4a.htm) and ExpressDB (http://arep.med.harvard.edu/ExpressDB/ExpressDB.v102.help.htm). These databases are mostly repositories for microarray data that allow storage and retrieval of raw data. In addition, ArrayDB is a laboratory information management system. Stanford Microarray Database (SMD) (http://www.dnachip.org/) stores raw and normalized data and their images. It also provides interfaces for data retrieval, analysis and visualization.

GeneCards project (http://bioinformatics.btk.utu.fi/genecards/), initiated at the Weitzman Institute of Science, aims to integrate all biomedical information related to human genes and their products into one web-accessible knowledge base. It collects and organizes the information retrieved from more than thirty five databases.

Kyoto Encyclopedia of Genes and Genomes (KEGG) (http://www.genome.ad.jp/kegg/) is an effort to computerize current knowledge of molecular and cellular biology in terms of the information pathways that consist of interacting molecules or genes and to provide links from the gene catalogs produced by genome sequencing projects. The KEGG project is undertaken by the Bioinformatics Center, Institute for Chemical Research, Kyoto University. It provides graphical and textual information on biochemical pathways as well as analysis tools.

There are currently more than 335 biological databases presented by Baxevanis, 2002. Although the international scientific community actively works on integrating all available biological information, it is not an easy task. Developing analysis tools and visualization techniques is another area of active research. In addition to the efforts of standardization of data formats, there is still the need to bring all these aspects of gene studies together in applications encompassing various gene data, analysis tools and visualization that can make interpretation of the results intuitive and easy to grasp. The application shown in this thesis is a small step towards facilitating the researcher in getting better insight from the data.

CHAPTER 2

SPECIFIC DESIGN OF THE SOFTWARE APPLICATION

As detailed in the introduction, currently a widely used approach in addressing biological questions is the microarray experiment. A microarray experiment is performed to study gene expression levels in different cells and tissues, at different stages of development. It could facilitate drug target discovery, to highlight differences between control and diseased cells and assist in patient diagnostics. Microarray data are being increasingly stored in public databases for easy access to any investigator and were reviewed in Chapter 2. A multitude of analysis techniques including data mining, statistical and artificial intelligence methods are applied to extract knowledge from the data. Some of these techniques that are appropriate for the current project, such as the clustering techniques, are outlined in the introduction section. Since the analysis includes dealing with vast amounts of numerical data, it is difficult for humans to comprehend them easily. Therefore, efficient graphical representations are needed for a fast and intuitive captioning of the data by the researchers. A dendrogram is one efficient way to visualize microarray data that have been grouped according to the similarity of their gene expression levels. Proposed and implemented is an application which adds functional graphical information to the results of a clustering analysis of microarray data. The aim of the application is to annotate with keyword information the genes as they are presented in the dendrogram after the hierarchical clustering has been performed. Upon selection of a keyword from the menu, the program attaches a color-coded icon to the genes in the dendrogram which contain that keyword in their biological database record. A general view which outlines the place of the current project among the available biological databases and analysis tools is presented in Figure 2.1

The more general problem can be disintegrated into several smaller problems. First, an organism or organisms which genes will be studied must be chosen. Then a biological database to be used for the extraction of the annotation information must be selected. One must decide whether the extraction is going to be dynamic at the time of the gene annotation or a local database with the pertinent information is to be designed and kept. If a local database is maintained, a DBMS has to be chosen. The method for by which the information pertaining to keywords and, possibly other information to be kept in the local database, is going to be extracted, parsed and used, must be selected. If a local database is used, it has to be connected to the application using the keyword information in the dendrogram.

2.1 Yeast as a Model Organism

Saccharomyces cerevisiae model organism was used for several reasons. *Saccharomyces cerevisiae* (Baker's yeast or yeast) is a simple eukaryotic organism. Its genome consists of only 6881 genes. It is widely used as a eukaryotic microorganism model for biological studies. The complete sequence of its genome was elucidated in 1996. Ever since, it was being used as a reference towards the sequences of human and other higher eukaryotic genes. Furthermore, since many genes are functioning in such fundamental and conserved processes such as cell cycle and growth, the yeast organism is used in complementarity assays where the yeast genes are replaced with their highly homologous sequences from human as well as other higher eukaryotes to provide insight into their

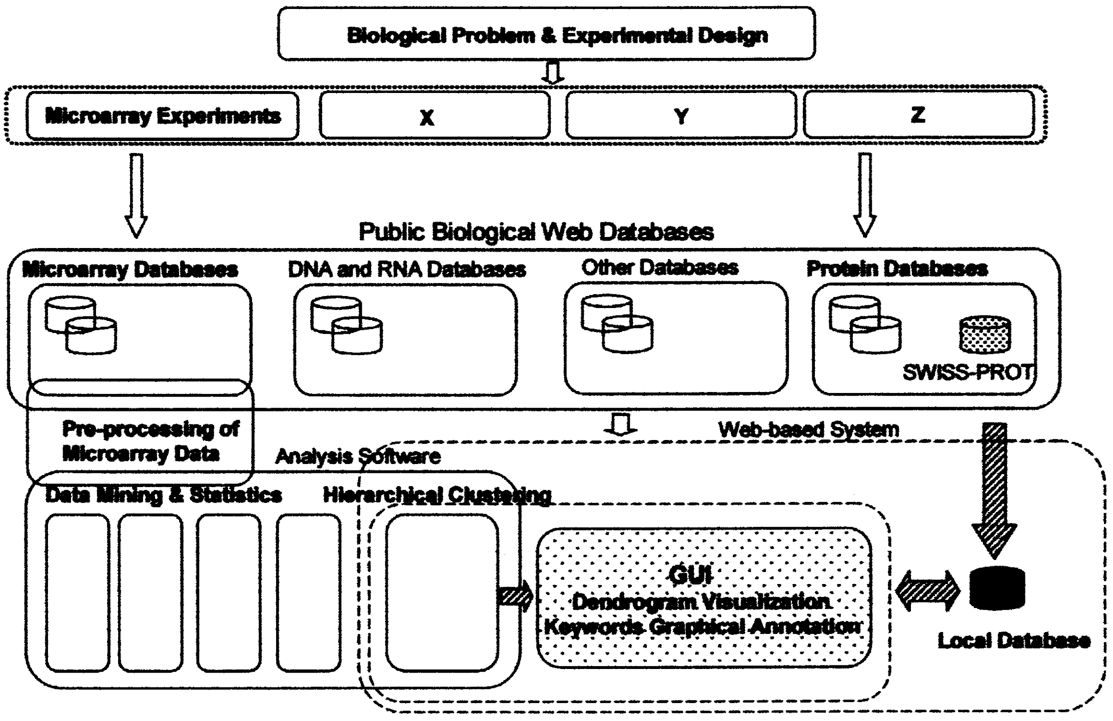


Figure 2.1 The diagram illustrates where the developed database and the application stand in relevance to the other available resources. The shaded areas and the shaded arrows in the lower right corner indicate the steps performed in this project. The dashed line outlines the system that could be developed in the future to include the analysis of microarray data, connectivity to the major microarray depositories, data from more biological databases and web-accessibility.

functions. Moreover, in the last few years microarray studies have become very popular and the team of P.O. Brown at Stanford University devised a protocol to make the appropriate tools low cost and, consequently, to accommodate the entire set of yeast genes to one chip. Thus, the whole network of yeast genes can be monitored for changes in expression profiles in varying biological conditions and analyze the gene functions and their inter-dependence.

2.2 Extraction of Keywords Information

Many publicly available biological databases exist that contain a wealth of information useful to the research community. GenBank was among the earliest to be established and it contains the most complete collection of gene sequence and sequence related information as well as access to protein sequence and protein structure data. GenBank provides links to other public resources too. The National Center for Biotechnology Information (NCBI) in cooperation with the European Molecular Biology Laboratory (EMBL) and the DNA Database of Japan (DDBJ) maintain sequence data from every organism, source and type of DNA. On the other hand, SWISS-PROT, PIR, and PRF are examples of protein databases. More specialized databases containing 3D protein structure information (Protein Data Bank), post-translational modifications (RESID), biochemical pathway data (KEGG, WIT), gene expression data and databases dedicated to a specific organisms are also available. For the current application, functional annotation information about yeast genes was extracted from a publicly available database.

The biological database of choice for this project was SWISS-PROT Protein Knowledgebase (http://www.ebi.ac.uk/swissprot/Information/information.html). It is maintained collaboratively by the Swiss Institute for Bioinformatics (SIB) and the European Bioinformatics Institute (EBI). The SWISS-PROT Protein Knowledgebase is an annotated protein sequence database established in 1986 and has been denoted as the most accurate and timely updated protein database. It provides annotations such as the description of protein function, domains structure, post-translational modifications, keywords, variants, etc. Cross-references are made in SWISS-PROT to multiple other databases among which is the Saccharomyces Genome Database (SGD) at Stanford University (http://genome-www.stanford.edu/Saccharomyces/). The SWISS-PROT Protein Knowledgebase was chosen for the extraction of the keyword information because it is a curated protein sequence database that provides a high level of annotation, a minimal level of redundancy and high level of integration with other databases. The SWISS-PROT is accompanied by TrEMBL, a computer-annotated supplement to SWISS-PROT. The TrEMBL database contains the translations of all coding sequences (CDS) present in the DDBJ/EMBL/GenBank Nucleotide Sequence Database and also protein sequences extracted from the literature or submitted to SWISS-PROT, which are not yet integrated into SWISS-PROT. The TrEMBL records for the yeast CDS were also used in the annotation in the tool presented. The number of SWISS-PROT yeast entries is 4880 and the number of TrEMBL yeast entries at the time of this thesis preparation was 2004.

In order to extract the useful functional information, the data from the public databases had to be parsed and Perl language was chosen. This task could be performed by a program written in other computer languages. However, Perl is a platform-independent language especially suitable for work with strings (Schwartz and Christiansen, 1997). If C++ were used, it had to be written for a particular platform and either be rewritten to be used for another platform, or restricted to the current one. Perl was chosen to parse the original data files from SWISS-PROT because of its ease of use, flexibility and efficiency. Perl has flexible syntax, takes care of common tasks, such as memory allocation, and it has powerful constructs. A program in C++, for example, would require more error checking, although it may be optimized to be faster. However, since the parsing is not a process done frequently, the speed is not the biggest concern. In order to sort through and analyze the available yeast data at SWISS-PROT, a Perl script was written because of Perl's strong pattern finding capacity, especially strings in text. In contrast, C++ does not have the pattern matching or regular expression usage to the extent that Perl offers, and that is particularly important when working with strings. Thus, programming problems that would require many lines of code in C++ or Java could be solved in a few statements in Perl (Gibas and Jambeck, 2001). The next candidate of choice would be Python. Although Python is less known than Perl, however, it is a fully object-oriented language which has many useful contributed code and libraries related to biological data. Perl is highly portable and free for everybody. In addition, many useful modules relevant to molecular biology problems have been developed and are freely available through the BioPerl project (http://bio.perl.org/). These modules could be integrated easily into the existing code as needed.

Therefore, a Perl script was written which parsed the available yeast data from the SWISS-PROT downloaded file and was used to create the specific files needed to populate the MySQL database tables. It extracted the information about primary and secondary account numbers of the genes, the gene name, description and keyword information to be used for the annotations in the dendrogram of gene expression clustering.

2.3 Relational Database Design and Development

Instead of a dynamic retrieval of the yeast annotation information from the Web, an approach to build a local database was preferred in favor of the speed that can be achieved with a local database solution. As means of storing and retrieving the yeast gene information, a relational database was designed and built and later queried according to the user specifications.

A database management system (DBMS) was chosen versus a flat file system. It provides data independence from data representation and storage, and efficient data storage and retrieval. In addition, DBMS also ensures data integrity and security as well as concurrent access and crash recovery, although these features are not needed for the current project. It also reduces the application development time (Ramakrishnan and Gehrke, 2000). A database management system is a complex software optimized for certain kinds of workloads, of which answering a complex query is especially valuable for the currently described application system. Most DBMS today use the relational model where the high-level data description construct is a relation, or a set of records. While many databases that use hierarchical, network, object-oriented or object-relational models, the dominant one is the relational data model, used in systems such as DB2, Informix, Oracle, Sybase, Access, MySQL and other. The Structured Query Language

(SQL) is a subset of the Data Manipulation Language (DML) and provides constructs to insert, delete and modify data in the database. The ease of expressing queries in SQL has played a major role in the success of the relational database systems.

The choice of relational database used was based on the cost, availability, usability and performance. Oracle, Sybase and DB2 are heavily used in industry settings. However, they are prohibitively expensive. In contrast, anyone can download the MySQL software from the Internet and use it without paying anything. For the current project only a relatively small number of records and tables were needed. Therefore, the industrial strength of Oracle or Sybase that could maintain a lot of data in exchange for a high cost was not a priority.

It is hard to find solid performance data to choose among competing technology in the database space, especially since the database vendors use no-benchmarking clauses in their license agreements to block publication of benchmarks that they disapprove. A recent article in eWeek online magazine (Dyck, 2002, http://www.eweek.com/print_article/0,3668,a=23115,00.asp) reports on a benchmark test carried out by eWeek and PC Magazine. Performance data for the latest versions of five major server databases was compared. IBM's DB2, MS SQL Server 2000, MySQL 4.0.1 Max, Oracle Corporation's Oracle9i and Sybase were compared on the same hardware platform. Drivers, memory tuning and database design issues were the three factors that had the most impact on performance. The Oracle and MySQL drivers had the best combination of a complete Java Database Connectivity (JDBC) feature set and stability. MySQL 4.0.1 Max had a great performance mostly due to a new, extremely fast in-memory query results cache, not available with any of the other databases. If the text of an incoming query has a byte-for-byte match with a cached query, MySQL can retrieve the results directly from the cache without compiling the query, getting locks or doing index accesses. This feature could be especially useful in the presented project, where tables are loaded once and queried, and don't have transactions with updates, which clear the cache. The MySQL Database Server is very fast, reliable, and easy to use. The MySQL Database Software is a client/ server system that consists of a multi-threaded SQL server that supports different backends, several different client programs and libraries, administrative tools etc. There is a large amount of contributed MySQL software available. Other advantages of MySQL are that it works on different platforms, it has a wide range of programming interfaces (APIs) such as C++, Java, and Perl, and it handles large databases (DuBois, 2000). So, because of its high performance and free availability to the user, MySQL was the DBMS of choice. Server version 4.0.4-beta-max-nt was used. Since the company does not have its own driver, Mark Matthew's MySQL JDBC driver version 1.3, the same driver tested in the benchmark study done by eWeek and PC Magazine, was used.

An Entity-Relationship (ER) Diagram in Figure 2.2 displays the local yeast ER database schema and shows the data graphically in the form of entities, relationships and attributes. For the local MySQL database, the accession number of the yeast gene from SWISS-PROT or TrEMBL was used as a unique identifier of the gene entity. Numerical keyword ID was the key for the Keyword Information entity. The rest of the entities had only one, and therefore, key attribute.

The yeast database consists of five tables. Table ac_descr contains all the accession numbers and the relevant description of the gene with that accession number.

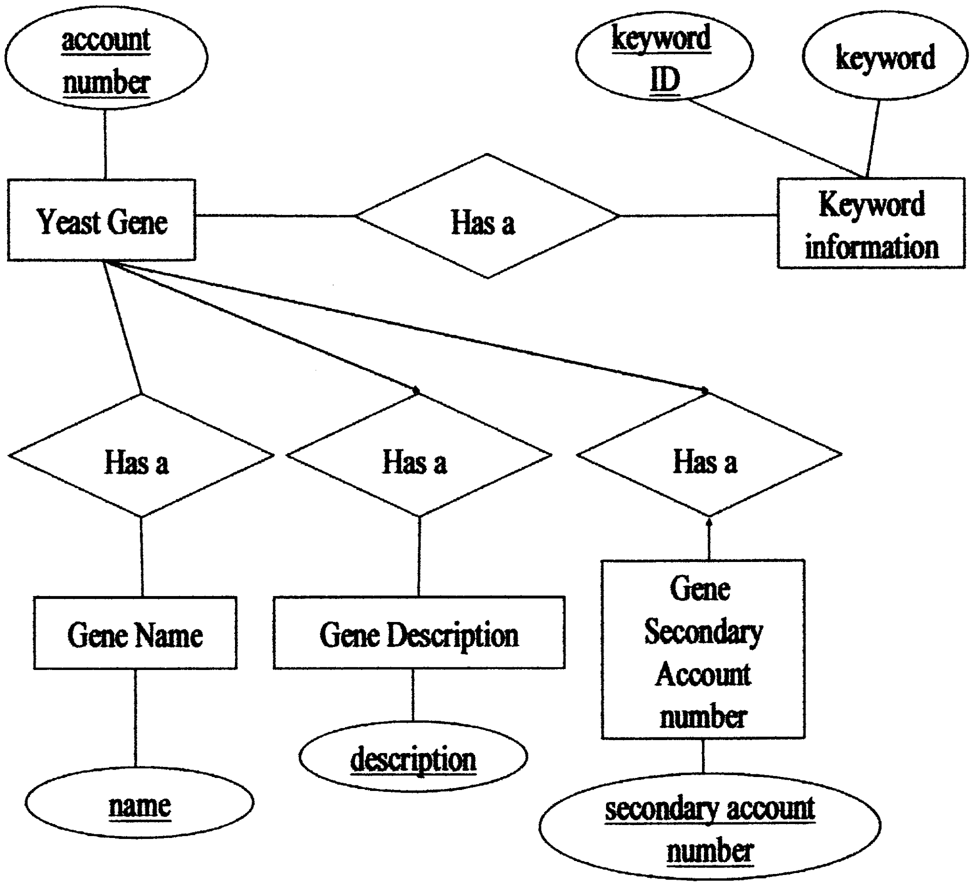


Figure 2.2 The figure represents an ER diagram of the yeast keywords database used to annotate the genes in the dendrogram. The entities are shown as boxes, attributes as ovals and relationships as diamonds. The key attributes are underlined. The four relationships and the "Keyword Information" entity were developed into database tables.

Some genes have a secondary number in addition to the primary, so the second_ac table was created. The second_ac table contains all primary accession numbers and their corresponding secondary accession numbers for a fraction of the genes that have both primary and secondary accession numbers. The ac_name table contains all the possible names for a gene with a particular accession number. The keywords associated with the genes were extracted while reading the annotations for each gene from the SWISS-PROT file. A keyword identification number (id) was assigned to each newly encountered keyword that was inserted in the database and so an index was created. It is important to note that the order of keywords is not alphabetical but in order of their appearance in the original data file. The table kw_id stores all the ids and their corresponding keywords. After each keyword was assigned an id, it was possible to associate each accession number with a keyword id(s) that are encountered for that accession number. This information is stored in the ac_kw table. With this design, just one read of the original data file is necessary to create all the necessary output files used to populate the database tables. The tables in the database are described in Appendix C.

2.4 Java Application for Graphical Functional Annotation

Java programming language was chosen mostly because of its platform independence, object-oriented design and ease of development when constructing a graphical user interface (GUI). As an alternative, C++ could be used. For example, Eisen's Cluster software was written in C++. However, C++ is platform-dependent and the program either had to have separate versions for the different platforms, or it had to be restricted to a particular one. The graphical nature of the application implies complicated components.

Since Java has a rich library of pre-defined classes, it makes it easy for the programmer to derive custom solutions based on the existing classes. Libraries are also available for C++, but these might not be incorporated in the language and may not be free. In addition, C++ could be more efficient, but is not so friendly to the developer. It could take a longer time for development, and require more error checking and more care when working with memory addressing. Java, in contrast, does not have pointer arithmetic. Most likely more code would be generated by using C++ and, therefore, make it more difficult for future changes to be incorporated. In addition, the built-in data structures as ArrayList, HashMap, TreeMap and Strings are very useful for the current application and for a shorter development time. Currently, Java is a more popular choice for GUI development, and if in the future someone would like to modify the code, programmers would likely have more experience developing GUI in Java. Based on the above considerations, Java was a natural choice.

The application takes as an input a .cdt file produced by the Cluster application written by M. Eisen (Eisen et al., 1998) at Stanford University, available at no cost for download on the web (http://rana.lbl.gov/EisenSoftware.htm).

A freely available code for an application performing tree visualization of clustering results, based on Eisen's TreevIew software, was adopted and modified to support the current project (Kavcic & Zupan, 2001). Annotation functionality code is developed and incorporated in the application. This way, a user starts with a .cdt file of numerical clustering results, displays the dendrogram according to his preferences either as a whole image, or as an enhanced part of the image showing the gene names. An Annotate Menu opens up a GUI that incorporates all the keywords in an alphabetical order and the available colors. It allows the user to select the desired keywords and colors in which he wants the annotations to be displayed. When the user specifies his requirements for the gene annotation, the program connects to the local MySQL database and performs the specific query, and returns the results. The information is then converted into a color-coded icon displayed next to the gene name in the dendrogram.

A flow diagram detailing the specific stages followed to design the application and the steps the user has to follow to generate the input file for the presented application is shown in Figure 2.3.

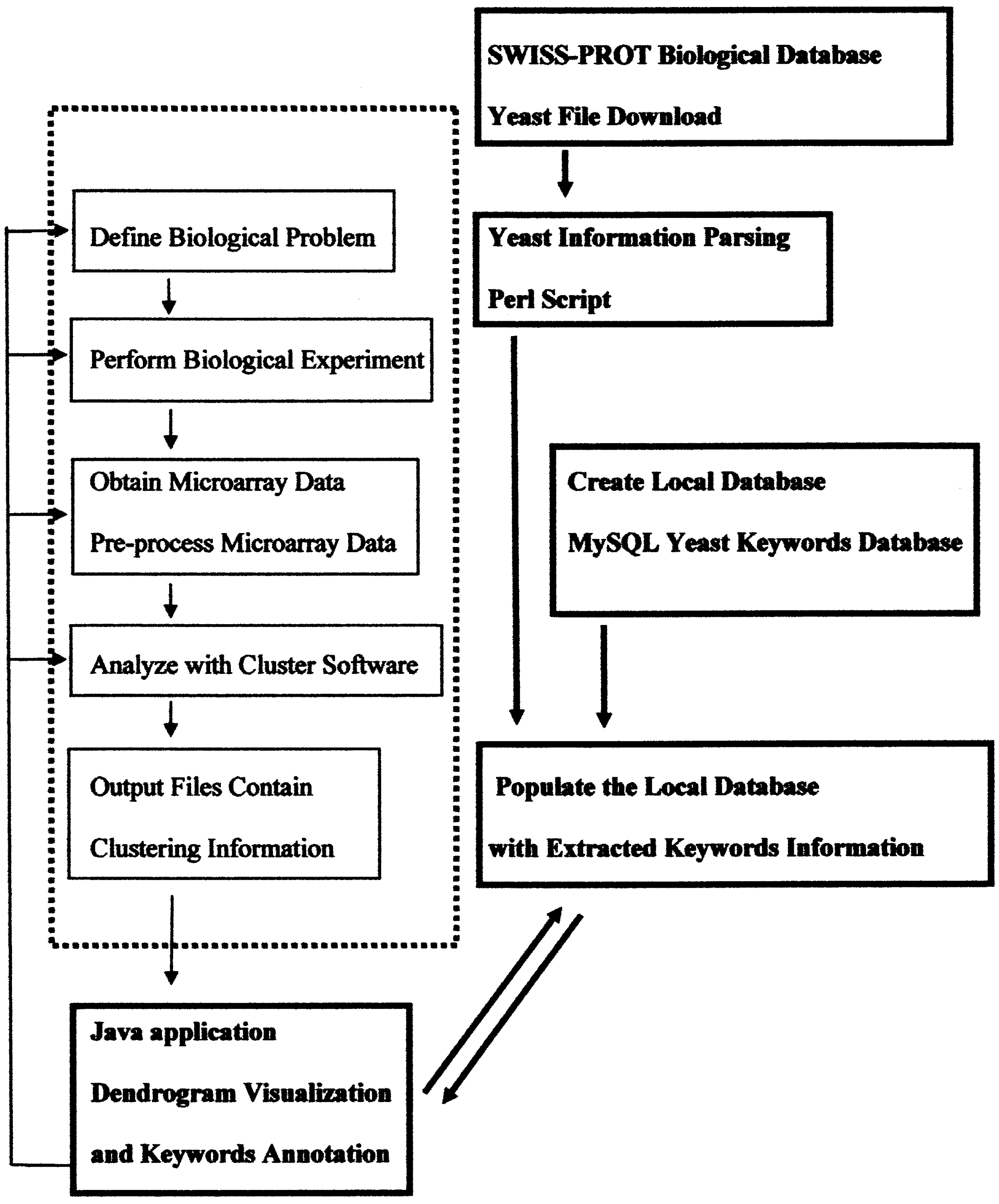

SWISS-PROT Biological Database
Yeast File Download
Yeast Information Parsing
Perl Script
Define Biological Problem
Perform Biological Experiment
Obtain Microarray Data
Pre-process Microarray Data
Analyze with Cluster Software
Output Files Contain
Clustering Information
Create Local Database
MySQL Yeast Keywords Database
Populate the Local Database
with Extracted Keywords Information
Java application
Dendrogram Visualization
and Keywords Annotation

Figure 2.3 A flow diagram illustrating the process of graphical annotation of the microarray gene expression data. The dashed area shows steps to be performed before using the program and the tasks in bold illustrate the design and implementation of the application.

CHAPTER 3

RESULTS

Java application tool was developed to graphically annotate and visualize the gene expression clustering results of microarray experiments. The work was divided among three main tasks: writing a Perl script to parse the SWISS-PROT keyword and related information, designing and implementing a local relational database in MySQL and building a graphical user interface with its supporting logic for the annotate menu of the application which queries the database and displays the keyword annotations for the gene graphically in a color-coded scheme.

1.1 Implementation Highlights

The data necessary for the annotation of the yeast genes was obtained from SWISS-PROT knowledgebase. A text file containing yeast accession number, description, gene names and the associated keywords for all the *S. cerevisiae* (Taxonomy ID 4932) genes was downloaded from SWISS-PROT (http://genome-www.stanford.edu/Saccharomyces/lists_tables.html).

A Perl script was written that parses all the information from the SWISS-PROT original file (Perl Kit, Version 5.6.1). The script reads the input file just once, parses it and creates five output files which consist of insert SQL statements to fill the records in the five tables of MySQL yeast_kw_db local database that holds information to be used for the annotation of the genes in the dendrogram. A bat file (Appendix B.3) was created which, upon execution, connects to the MySQL Server and loads the output files from the

Perl script into the appropriate tables of the database. The Perl script Generic.pl is accompanied with by comments and is attached in Appendix B.1 along with the list of the 333 extracted unique keywords used for the yeast genes annotation (Appendix B.4).

A simple database consisting of five tables was created using MySQL DBMS to hold the information of gene primary and secondary accession numbers, gene descriptions, gene names and keywords needed for the annotations. The database table's specifics can be found in Appendix C.

The annotation functionality was implemented using Java JDK 1.3 and JBuilder 7.0 Personal Edition. The FreeView program (http://magix.fri.uni-lj.si), which is a Java implementation of Michael Eisen's TreeView software (http://rana.lbl.gov/EisenSoftware.htm), was used as a skeleton for building additional functionality by the described application. The code was adopted and modified to accommodate the annotation process by adding an extra menu "Annotate". The application makes a JDBC connection to the local MySQL database to query it according to the user's specifications. The returned results are displayed in a graphical color-coded mode next to the genes containing the chosen keywords. A specific example of using the program and the specifics of the followed steps are detailed in section 1.2 of this chapter.

Loading of a file with the appropriate extension .cdt activates the "Annotate" menu (Figure 3.1). To demonstrate the graphical user interface of the "Annotate" menu, a screenshot is shown (Figure 3.2). The user is presented with alphabetically arranged tabs for 26 letters representing all the keywords. A tab "Other" is included to accommodate the keywords starting with numbers. Upon clicking on a tab with a particular letter, the

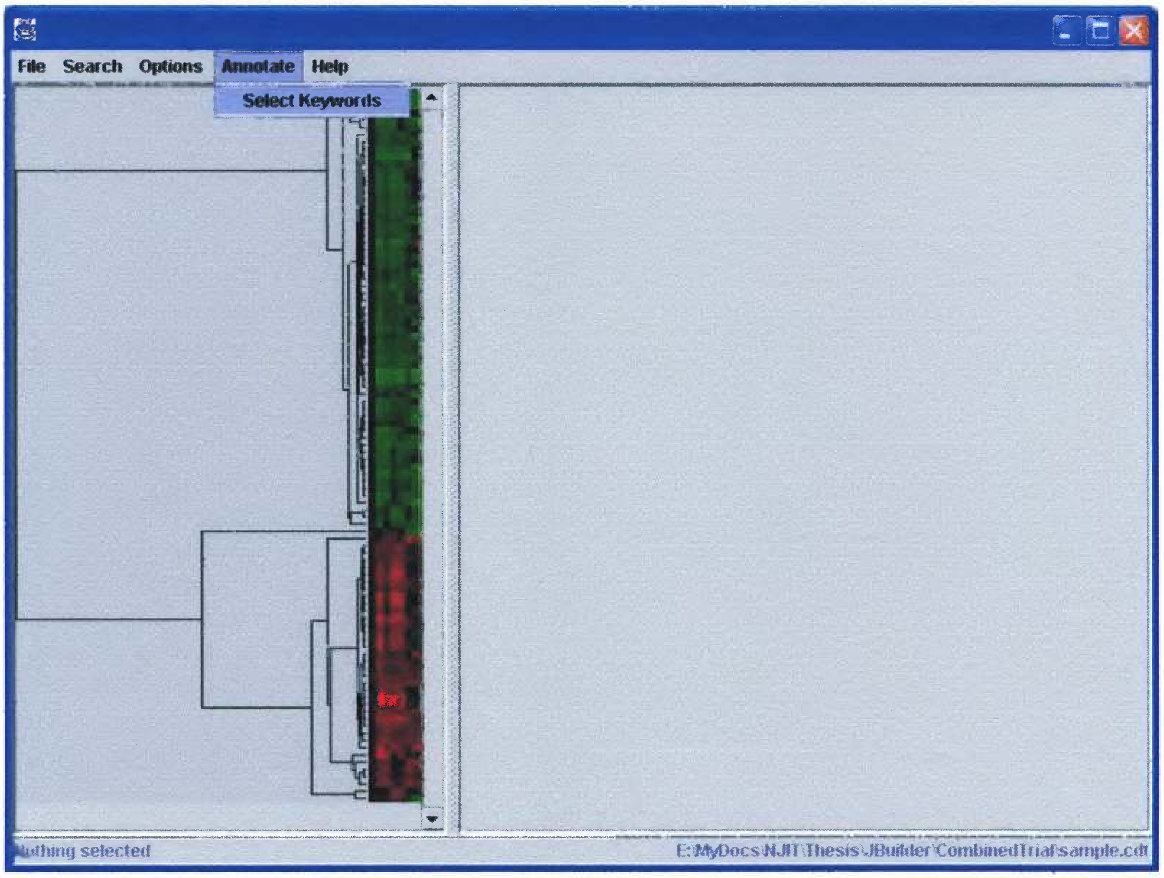
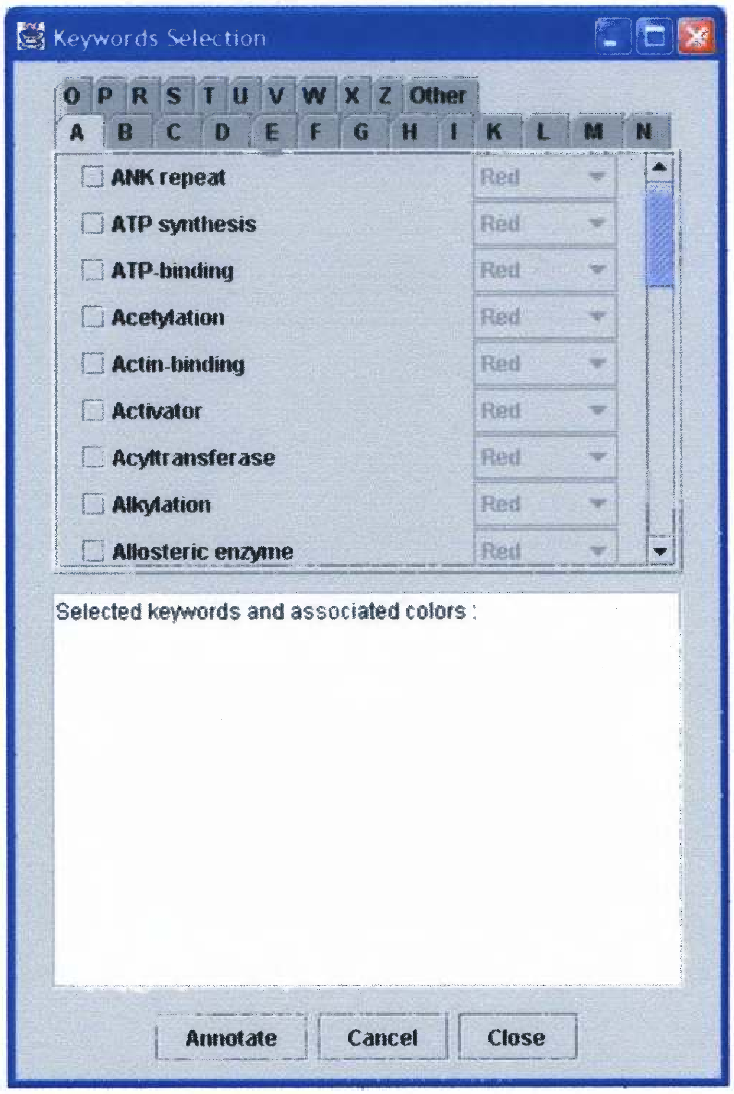


Figure 3.1 This figure represents the main frame of the java application. Upon loading the file, the dendrogram is displayed and the Annotate menu activated. The Annotate menu contains a Select Keywords menu item which will allow the user to select the desired keywords for the graphical annotation of the yeast genes in the dendrogram.



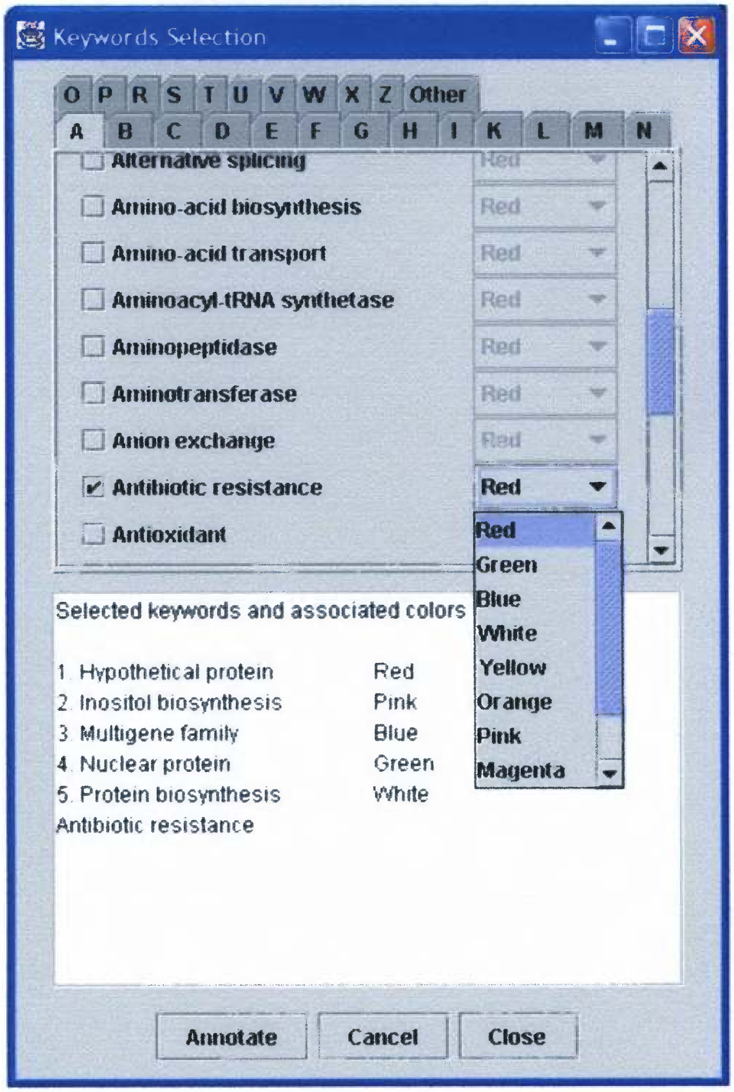
☐ ANK repeat	Red
☐ ATP synthesis	Red
☐ ATP-binding	Red
☐ Acetylation	Red
☐ Actin-binding	Red
☐ Activator	Red
☐ Acyltransferase	Red
☐ Alkylation	Red
☐ Allosteric enzyme	Red

Selected keywords and associated colors :

Figure 3.2 This figure shows the Keyword Selection frame. The keywords are presented in alphabetical order, where each letter tab represents all the keywords starting with that letter. The "Other" tab represents keywords starting with numbers. The scroll bar allows the users to see all the keywords starting with a particular letter. The text box area records and displays the user's choice of keywords and colors.

keywords starting with that letter are displayed as a vertical list checkboxes. A scroll bar is provided, in case all of the keywords cannot be viewed simultaneously. Next to each checkbox is a combo- box that holds the available colors for annotations. The combo-box becomes active after the user has selected its corresponding keyword checkbox (Figure 3.3). After a keyword and the color in which the user wants it displayed are selected, the textbox below the list of keywords records the current selections. Up to ten keywords can be selected at a time to fit into the limited space next to the clustered genes and to avoid cluttering the image.

Each keyword is presented as a color-coded square icon next to the gene name in the enhanced part of the dendrogram. For clarity, a legend showing the correspondence between a keyword and its colored icon is drawn next to the tree image. To create the GUI of the Preferences of the Annotate menu, GridBagLayout was used extensively because of the multiple components that had to be accommodated. After the user has clicked on the Annotate button, he has to select a part of the dendrogram he wants to see annotated. Upon clicking on a branch, the genes included in that branch are displayed in the right-hand side with the gene names added. Next to the gene names are displayed colored squares. Each colored icon corresponds to a keyword according to the selection the user has made. Each gene can potentially have all ten keywords, so a spot for each colored square is reserved. If a keyword does not describe a particular gene, its spot is left blank.



Selected keywords and associated colors

1. Hypothetical protein	Red
2. Inositol biosynthesis	Pink
3. Multigene family	Blue
4. Nuclear protein	Green
5. Protein biosynthesis	White
Antibiotic resistance	

Figure 3.3 This figure illustrates the process of keyword and color selection. It allows the user to choose a keyword by clicking on a check box. Upon selection of a keyword, the corresponding color menu is activated and the user can choose a color in which the keyword is to be represented graphically. Up to ten keywords could be selected. The text box displays the keywords sorted in alphabetical order.

1.2 Annotation Process Example

To demonstrate the application, a demo file sample.txt was clustered with the Cluster program. The output files sample.cdt and its companion sample.gtr, created by the Cluster software, were used as an input for the dendrogram visualization and the graphical keyword annotation. The files contain clustering information about 99 yeast genes tested in seven different conditions. The sample.cdt file was loaded into the java application and its dendrogram displayed (Figure 3.1). Upon clicking on the Annotate menu, the user is presented with a keyword selection frame (Figure 3.2). The process of selection of keywords is illustrated on Figure 3.3. The user first selects a keyword by clicking on a check box. That activates the combo box and allows the user to choose a color in which he wants to denote the particular keyword. In a similar way, ten keywords with the appropriate colors were selected for the genes in the demo file. Two sub-clusters of the dendrogram, corresponding to down- and up-regulated genes with similar expression were selected by clicking on a branch of the dendrogram and the clustered genes annotated. The selected sub-clusters are colored in pink and shown on the left pane (Figure 3.4 and Figure 3.5). The resulting image with enhanced gene expressions, gene names and the color-coded annotations is shown in the right panes of the Figures 3.4 and 3.5. In the particular example, the user could instantly see which genes have their 3D-structure solved or which genes are described as hypothetical proteins. As it could be expected, different patterns of keyword annotation are present for the two selected clusters in Figure 3.4 and in Figure 3.5. For example, certain keyword icons can be seen only for some of the down-regulated genes and not for the up-regulated ones. This can point out to some of the common characteristics of the genes clustered together and

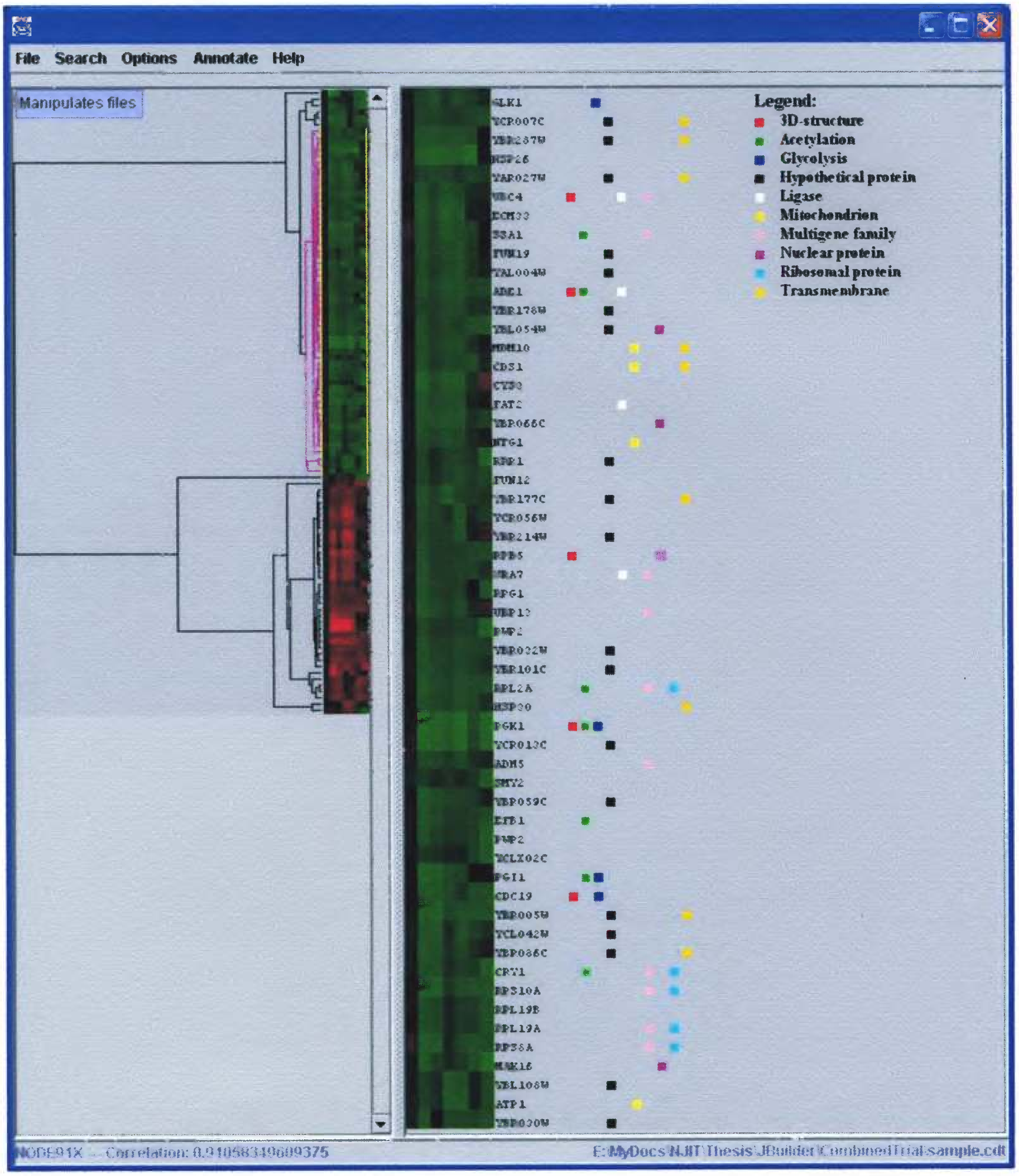


Figure 3.4 This figure represents the color-coded functional annotation of the down-regulated genes selected by the user. The legend shows the correspondence between keywords and colors. The colored squares in a row indicate that the particular gene can be described with keywords denoted by the colored squares.

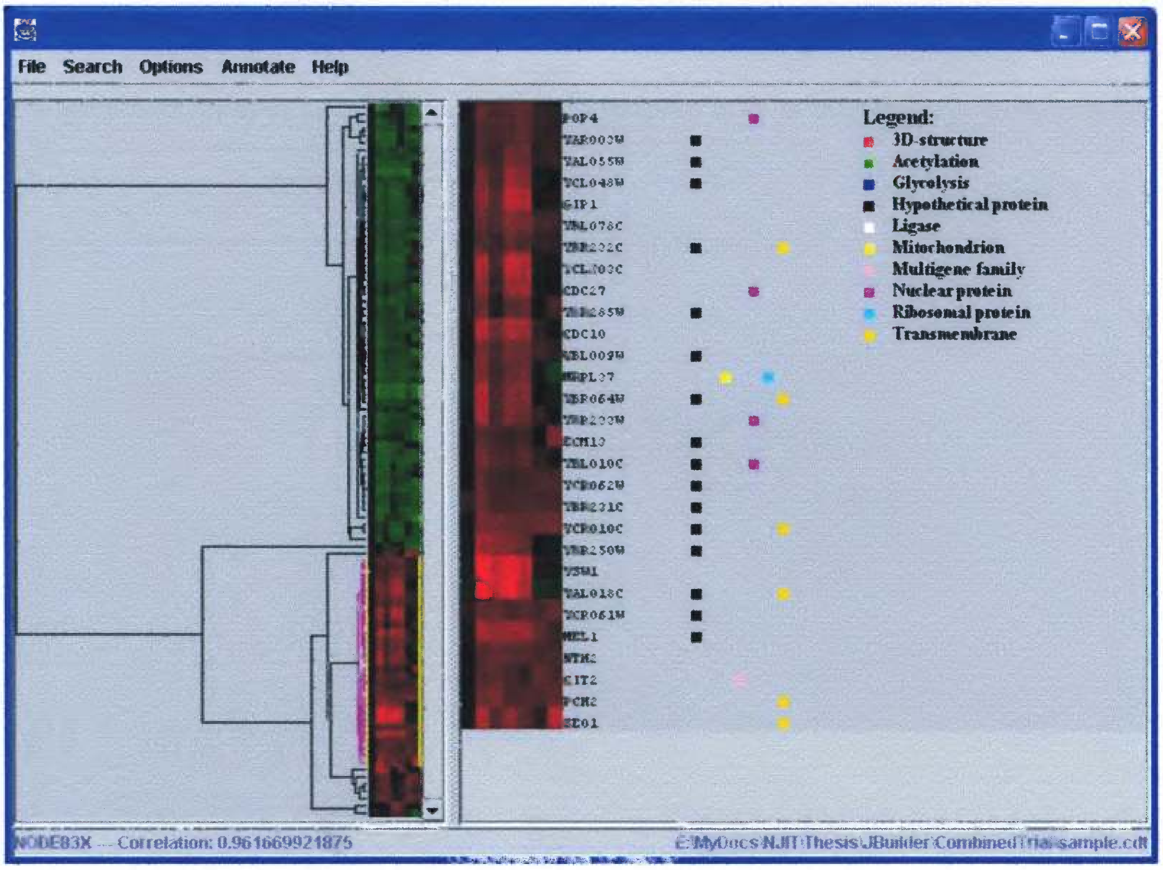


Figure 3.5 This figure represents the graphical color-coded functional annotation of the up-regulated genes in the experiment. After the user has selected up to ten keywords and their corresponding colors, he could click on the part of the dendrogram containing the yeast genes he wants to see annotated. The right part of the figure shows an enlarged view of the selected branch with additional gene name information and color-coded keyword annotation. The legend provides the correspondence between the keywords and their colors.

absent in the other clusters. It could save the researcher a manual look up that could be a tedious and time-consuming procedure. It could also lead to an insight about the particular question being studied.

The presented application is a tool that adds a layer of biological knowledge in a graphical way that is easy to grasp. The usefulness of the annotations will ultimately depend on the particular experimental set up, clustering conditions and the interests and expertise of the researcher as well as the biological problems addressed.

CHAPTER 4

CONCLUSIONS

4.1 Goals Achieved

A Java bioinformatics graphical gene annotation tool was built which allows the display of yeast gene expression clustering results along with gene keyword information in a color-coded scheme. The display and annotation tool developed helps to bridge the gap between the raw data gathered from microarray gene expression experiments and their interpretation by the researchers interested in gene function, gene interaction and co-regulation, in cellular response to drugs etc. Although many useful tools exist, most of the time it is through manual look-up that the scientist establishes the usefulness of the gene clustering results. He has to spend a lot of time researching the functional information available on the web in separate databases for each individual gene and integrate the useful information in the process. It is a tedious and time-consuming process that can be emotionally drowning and not very efficient. A tool which allows the integration of the available functional data and the results from experimental data analysis techniques, and allows intuitive representation, will potentially speed up the process of hypothesis testing and generation in biological research.

Keywords describing gene function and biological processes involved were chosen to graphically enhance the microarray clustering results with functional information. Perl script was written to extract the keywords and other pertinent information from SWISS-PROT. It can be easily adapted for other uses concerning other organisms or other available information. In the same line of thinking, the database that was created can be taken as a separate module and can be easily modified to accommodate more gene features, to include more organisms, or be adapted for use in another bioinformatics application.

4.2 Future Recommendations

Although useful, the described application can be further developed to enhance its functionality, usability or to address other needs a researcher might have.

A useful way to further enrich the program could be to automatically show the most common keywords for a selected branch of the dendrogram. This will point out the common characteristics of the genes in the selected branch and help get an insight of their relationships. The implementation would use the same database with a different pattern matching logic. The annotation would be automatic and would not require input from the user. In addition, it would be useful if a gene name is made as a hyperlink to different databases of choice, so the user can look up any information he needs from them right away.

Alternatively, the application could be further developed to also show only the keywords that describe a particular branch of the dendrogram and not other branches. This will also give an idea to the investigator about the specific functions or processes that characterize only a particular set of genes. It could be added as an additional option for annotation.

The current application is a limited–scope effort and could only work with the yeast model organism. Since the integration of biological information from different organisms greatly increases quantitatively and qualitatively our understanding of how we function, it would be necessary to include available information about other organisms too, and, especially, human data. This would mean that other databases would have to be accessed, information extracted and included in the local database. As the complexity increases, the design choices for the application could also change. In addition, a key improvement of the program could be make the application Web-accessible to allow many researchers use its functionality simultaneously. Regular updates of the local database would be necessary to provide reliable information to the scientific community.

As mentioned earlier, a large-scale systematic effort to annotate many organisms is under way. The Gene Ontology Consortium and GO project will provide a systematic way to describe molecular function, biological processes and cellular components. The gene ontology represents the information in a tree-like structure where the parent nodes provide a general description and the leaf nodes contain the specific terms. Since the GO knowledge is a more systematic and organized method to represent the gene information, a GO similar color-coded annotation could be a better alternative to the keywords annotation. It could provide different levels of generalization or specificity of the gene knowledge that could be controlled by the user.

One way of enriching the application could be to add graphical information displaying the different biochemical pathways in which a gene or a group of genes are participating. This is a natural way to represent gene-gene interactions or gene regulation. Although such maps were developed to some extent, the systematical representation of gene function is under way. Kyoto Encyclopedia of Genes and Genomes designed and maintained by Dr. Kanehisa and colleagues contains a large collection of graphical maps (http://www.genome.ad.jp/kegg/kegg3.html). The gene relationships represented are partially extracted from the biological literature by experts and partially deduced or computed from previously stored knowledge by using so called binary relationships (Kanehisa et al., 2002). KEGG makes the effort to incorporate all the available biological information in a graphical way in the form of maps. If included along with the dendrogram, this type of representation can greatly increase the efficiency with which a researcher can interpret the expression results. This would also be a step towards systems versus the reductionistic approach in biological research.

Since it is often thought that genes that share similar expression patterns are co-regulated, it could be really fruitful to look at the non-coding regions of the gene. These regulatory regions contain transcription factor binding sites, consisting of stretches of several nucleotides which mediate up- or down-regulation of the gene expression. It has been observed that genes with similar expression share these protein binding sites responsible for their regulation. However, the common appearance does not automatically mean common regulation, since different binding sites might act synergistically to produce a different response. Although comparing control region sequences and functional information is by itself a topic of a study, the available promoter information could be used to add another layer of functional information to researchers studying the gene expression profiling. Several studies coupling promoter analysis with expression patterns have been performed (reviewed by Dutilh and Hogeweg, 1999). It could be especially helpful if the information is presented in a graphical way. Flexibility could be achieved if the user is given the choice of matching a particular sequence pattern or choosing a transcription factor, or particular pathway and relating the results to the clustering patterns.

Although the starting point for this application was gene expression results, gene expression is just one approach from the multitude possible to study a cell or an organism. Thus, it should be viewed as a part of a big picture and not taken in isolation. In this line of thought, it should be mentioned that a platform for the integration of functional genomics with the scientific literature, named The BioKnowledge ™ Library (http://proteome.nih.gov:8000/may2001/garrels.html) has been established. Its idea is to summarize information on yeast functional genomics, such as genome-wide gene knockout, transcript profiling, microarray datasets, results from systematic two-hybrid screens, drug target discovery, and yeast proteomics together with yeast research found in the scientific literature. This database was designed to also incorporate similar data sets from other genomes in addition to the yeast data so to allow global interpretation, to analyze common protein functions, to explain common modes of regulation, and functioning of cellular pathways.

In summary, existing biological data must be integrated together and represented in an intuitive graphical way so that knowledge rather that information is extracted, hypotheses are generated or tested more efficiently and in less time. The tool presented is just a minute step in the process of integration and representation of biological data to achieve greater understanding and apply it in the quest of improving human health and quality of life.

APPENDIX A

JAVA FILES

This appendix contains the source code written in Java and used in combination with code adopted from another application to visualize and annotate yeast gene expression clustering. Statements used for debugging purposes are left in the file in case needed in the future and are commented. The triple dots in the files mean adopted code in the application.

```
package combinedtrial;

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.awt.image.renderable.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.JFrame;
import java.awt.Rectangle;
import java.lang.String;
import javax.swing.event.*;
import java.util.*;
import javax.swing.JMenu;
import javax.swing.JTabbedPane;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.sql.*;


public class MainFrame extends JFrame {
/**The frame for selection of the keywords and colors.*/
class KeywordsFrameOptimized extends JFrame {
  /**Create frame for selection of the keywords and colors.*/
  public KeywordsFrameOptimized()
  {
    //set the properties of the frame
    setSize(WIDTH, HEIGHT);
    setTitle("Keywords Selection");

    //get the screen dimensions
    Toolkit kit = Toolkit.getDefaultToolkit();
    Dimension screenSize = kit.getScreenSize();
    int ScreenWidth = screenSize.width;
    int screenHeight = screenSize.height;

    //open the frame at this location on the screen
    setLocation(ScreenWidth/4, screenHeight/4);

    Container contentPane = getContentPane();
    BorderLayout layout = new BorderLayout();
    contentPane.setLayout(layout);

    buttonPanel = new JPanel();
    buttonPanel.setVisible(true);

    //add a button to annotate with the chosen words
    annotateButton = new JButton("Annotate");
    buttonPanel.add(annotateButton);
    annotateButton.addActionListener(new
                                  ActionListener()
    {
      public void actionPerformed(ActionEvent  annotateEvent)
      {
        annotateSession = new Connect();
        annotateSession.fillColorArray();

        //close the annotation frame now
    dispose();
  }
});

//add a button for the user to cancel the annotation and empty the
tree and hash tables
cancelButton = new JButton ("Cancel");
buttonPanel.add(cancelButton);
cancelButton.addActionListener(new
                                ActionListener()
{
  public void actionPerformed(ActionEvent cancelEvent)
  {
    textArea.setText("Selected keywords and associated colors :" +
    "\n");
    totalSelectedKeywords = 0;

    for (keywordId = 1; keywordId < KwNumber; keywordId++)
    {
      //iterate through all the remaining entries of the hash map
      Set entries = selectedKeywordsAndColors.entrySet();
      Iterator iter = entries.iterator();
      while (iter.hasNext())
      {
        Map.Entry entry = (Map.Entry)iter.next();
        Object key = entry.getKey();

        if ((String)checkBoxes[keywordId].getText() == key)
        {
          //deselect all of the selected keywords
          checkBoxes[keywordId].setSelected(false);
          //System.out.println("Keyword" + keywordId + "  is
          deleted.");
          break;
        }
      }
    }

    //empty the tree and hash tables
    selectedKeywordsAndColors.clear();
    chosenColors.clear();
    totalSelectedKeywords = 0;
  }
});

//add a button for the user to close the window
closeButton = new JButton ("Close");
buttonPanel.add(closeButton);
closeButton.addActionListener(new
                              ActionListener()
{
  public void actionPerformed(ActionEvent  closeEvent)
  {
    //empty the hash table
    selectedKeywordsAndColors.clear();
    dispose();
  }
});

keywordsPanel = new JPanel();
lowerPanel = new JPanel();
```

```
lowerPanel.getPreferredSize();
upperPanel = new JPanel();
upperPanel.getPreferredSize();
middlePanel = new JPanel();

textArea = new JTextArea();
textArea = new JTextArea();
textArea.setEditable(false);

textArea.setText("Selected keywords and associated colors :" +
"\n");
textArea.setBorder(BorderFactory.createEtchedBorder());
Dimension sz = new Dimension(WIDTHTEXTAREA, HEIGHTTEXTAREA);
textArea.setPreferredSize(sz);
textArea.setVisible(true);
middlePanel.add(textArea);
contentPane.add(middlePanel, BorderLayout.CENTER);
//add the lower panel to the content pane
contentPane.add(lowerPanel, BorderLayout.SOUTH);

//add text area and button panel to the lower panel
lowerPanel.add(buttonPanel, BorderLayout.SOUTH);

//contentPane.add(keywordsPanel, BorderLayout.CENTER);
contentPane.add(keywordsPanel, BorderLayout.NORTH);

tabbedPane = new JTabbedPane();
Dimension sizeTP = new Dimension(WIDTHTABBEDPANE, HEIGHTTABBEDPANE);
tabbedPane.setPreferredSize(sizeTP);
tabbedPane.setTabPlacement(JTabbedPane.TOP);

panels = new JPanel[panelNumber];
tabPanels = new Component[panelNumber];
checkBoxes = new JCheckBox[KwNumber];
colors = new JComboBox[KwNumber];
keywordListener = new CheckBoxListener();
colorListener = new ColorAction();

//make a TreeMap (sorted HashMap) to hold key-value pairs (Keyword
text-color pairs)
selectedKeywordsAndColors = new TreeMap();
chosenColors = new HashMap();

// A hash table to hold all the keywords
yeastKeywords = new HashMap();
//an array list to hold all the letters used by the keywords
allLetters = new ArrayList();
allLetters.ensureCapacity(MAXLETTERS);

try
{
  //read the keywords from a file
  FileInputStream fin = new FileInputStream(
  "YeastKeywordsAlphab.dat");

  BufferedReader in = new BufferedReader(
      new FileReader("YeastKeywordsAlphab.dat"));

  String line;
  String kwrd;
```

```
int counter = 0;
String first = "";
String last = first;
int let = 0;
//number of letters in the array of letters
int totalLetters = 0;
while  line= in.readLine()) != null)
{
  //construct a StringTokenizer object that attaches to a string
  delimited by new line character
  StringTokenizer t = new StringTokenizer (line, "\n");
  kwrd = t.nextToken();

  first = kwrd.substring(0, 1);
  //System.out.println("First letter is :" + first);

  //if array size is less than two
  if  totalLetters == 0)
  {
    //add the first letter letter
    allLetters.add(new String (first));

    totalLetters++;
    last = first;
  }

  else if (totalLetters < 2 && (!(first.equalsIgnoreCase((String)
  last

  {
    //if the next letter is not already in the array
    allLetters.add(new String (first));
    totalLetters++;
  }

  else if (totalLetters >= 2)
  {
    int flag = 0;
    //if it is in the letters array list, ignore it; else add it
    to the letters array list
    for (let = 0; let < allLetters.size(); let++)
    {
      String temp = "" + let;
      //find the current letter's index and if it's not already in
      the array list
      if (first.equalsIgnoreCase((String)allLetters.get(let)) ||
      first.equals(temp))
      {
        flag = 1;
      }
    }
    if (flag != 1)
    {
      //add it to the array list
      allLetters.add(new String (first));
      totalLetters++;
    }
  }
  //retrieve the keywords into a hash table
  yeastKeywords.put("" + counter, kwrd);
  //increment the counter
```

```
    counter++;
   }
}
catch (IOException exception)
{
  exception.printStackTrace();
}

//for all the letters in the array list
for (int tabs = 0; tabs < allLetters.size(); tabs++)
{
  int kw = 0;
  //create tab panels
  tabPanels[tabs] = makeTextPanel( "" + allLetters.get(tabs) );
  panels[tabs] = new JPanel();
  setGrid(panels[tabs]);

  //for all the keywords
  for (kw = 0; kw < yeastKeywords.size(); kw++)
  {
    //get the first letter of the value
    String startsWith = ((String)yeastKeywords.get("" + kw)).
    substring(0,1);

    //if keyword starts with that letter, add the row to the panel
    if (startsWith.equalsIgnoreCase((String)allLetters.get(tabs)))
    {
      addRow (panels[tabs], (String)yeastKeywords.get("" + kw) );
      letterKws++;
    }
  }
  scrollPane = new JScrollPane(panels[tabs],
     JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
     JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
  tabbedPane.addTab(("" + allLetters.get(tabs)), null, scrollPane,
  "Keywords starting with this letter");
  setVisible(true);
}
//add the keywords panel to the tabbed pane
keywordsPanel.add(tabbedPane);

//add the rest of the keywords starting with a number in a tab
"Others"
tabPanels[allLetters.size()] = makeTextPanel( "Others" );
panels[allLetters.size()] = new JPanel();
setGrid(panels[allLetters.size()]);

//iterate through the remaining entries in the hash map
for (int rest = letterKws; rest < yeastKeywords.size(); rest++)
{
  addRow (panels[allLetters.size()], (String)yeastKeywords.get("" +
  rest));
}
JScrollPane scrollPane = new JScrollPane(panels[allLetters.size()],
    JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
    JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
tabbedPane.addTab(( "Other" ), null, scrollPane,
"Keywords starting with a number");
setVisible(true);
}
```

```
/**Create objects of type Constraints and GridBagLayout and set it to
be the layout manager for the panel
 * @param panel - the panel to which the layout will be set
 */
public void setGrid(JPanel panel)
{
  GridBagLayout layout = new GridBagLayout();
  panel.setLayout(layout);
  //panel.setSize(200, 600);
  constraints = new GridBagConstraints();
  row = 0;
}

/** Add keywords to the panel
@param panel the panel to which to add the components
@param keyword String - the keyword to be added
@param x int specifies the column position in the upper left corner of
the component to be added
@param y int specifies the row position in the upper left corner of
the component to be added
@param w int specifies how many columns the component occupies
@param h int specifies how many rows the component occupies
*/
public void addKeyword(JPanel panel, String keyword,
GridBagConstraints constraints, int x, int y, int w, int h)
{
  //first fill the the GridBagConstraints object for each component
  //and then add the component with the the constraints to the grid
  constraints.fill = GridBagConstraints.HORIZONTAL;
  constraints.anchor = GridBagConstraints.NORTHWEST;
  constraints.weightx = 0;
  constraints.weighty = 0;

  keywordId++;

  checkBoxes[keywordId] = new JCheckBox( keyword );
  checkBoxes[keywordId].addItemListener(keywordListener);

  constraints.gridx = x;
  constraints.gridy = y;
  constraints.gridwidth = w;
  constraints.gridheight = h;
  panel.add( checkBoxes[keywordId], constraints );
}

/** Add color JComboBox components to the panel with the corresponding
constraints
@param panel JPanel - the panel to which the components will be added
@param constr - the constraints to the component
@param x specifies the column position in the upper left corner of the
component to be added
@param y specifies the row position in the upper left corner of the
component to be added
@param w specifies how many columns the component occupies
@param h specifies how many rows the component occupies
*/
public void addColor(JPanel panel, GridBagConstraints constraints, int
x, int y, int w, int h)
{
  //set the constraints for the combo color box
  constraints.fill = GridBagConstraints.NONE;
  constraints.anchor = GridBagConstraints.NORTHWEST;
  constraints.weightx = 0;
  constraints.gridheight = 0;

  //now add the color combo box to the keyword
  colors[keywordId] = new JComboBox(new String[]
                                    {
                                    "Red",
                                    "Green",
                                    "Blue",
                                    "White",
                                    "Yellow",
                                    "Orange",
                                    "Pink",
                                    "Magenta",
                                    "Cyan",
                                    "Black"
  });
                                    colors[keywordId].
                                    addActionListener(colorListener);

                                    colors[keywordId].setEnabled(false
                                    );
                                    constraints.gridx = x;
                                    constraints.gridy = y;
                                    constraints.gridwidth = w;
                                    constraints.gridheight = h;

                                    panel.add( colors[keywordId],
                                    constraints );
}

/** Add components to the panel
@param panel the panel to which to add the components
@param c Component - the component to be added
@param x specifies the column position in the upper left corner of the
component to be added
@param y specifies the row position in the upper left corner of the
component to be added
@param w specifies how many columns the component occupies
@param h specifies how many rows the component occupies
*/
public void add(JPanel panel, Component c, GridBagConstraints
constraints, int x, int y, int w, int h)
{
  constraints.gridx = x;
  constraints.gridy = y;
  constraints.gridwidth = w;
  constraints.gridheight = h;
  panel.add( c, constraints );
}

/**A method to add a row containing keyword and color in the panel
 @param panel the panel to which the row is added
 @param keyword the keyword that is added to its panel
 */
private void addRow( JPanel panel, String keyword )
{
  addKeyword(panel, keyword, constraints, col1, row, width, height);
  addColor(panel, constraints, col2, row, width, height);
  row++;
}

/** This class is an inner class  which is covenient when writing
event-driven programs; it adds or removes a keyword the current
selection
and the text box; warns the user if the number of keywords exceed the
maximum
*/
public class CheckBoxListener implements ItemListener
{
  /**the interface must implement this method to carry out the
  functions of the class*/
  public void itemStateChanged(ItemEvent e)
  {
    //in case the user deselects the checkbox
    if (e.getStateChange() == ItemEvent.DESELECTED)
    {
      //remove it from the text area
      textArea.setText("Selected keywords and associated colors :" +
      "\n");

      //decrement the count the number of selected keywords up to now
      totalSelectedKeywords--;

      //find the keywordId of the keyword deselected
      for (keywordId = 1; keywordId < KwNumber; keywordId++)
      {
        Object source = e.getItemSelectable();

        //if proper keyword id
        if (source == checkBoxes[keywordId])
        {
          colors[keywordId].setEnabled(false);

          //if color in selectedKeywordsAndColors is different from
          NONE (if color has been already selected when checkbox
          deselected by user)
          //remember the key v in selectedKeywordsAndColors = keyword
          for that color
          String v = (String)checkBoxes[keywordId].getText();
          //retrieve an object (value) that corresponds to that key
          and compare it to null
          //to check whether color has been entered in the
          chosenColors tree map
          //System.err.println( " selectedKeywordsAndColors.get(v): "
          + selectedKeywordsAndColors.get(v));
          if (selectedKeywordsAndColors.get(v) != "None")
          {
            //System.err.println( " Color is different from null: " +
            selectedKeywordsAndColors.get(v));
            //delete that pair from the hash table (Holding colors and
            keywordws) since checkbox is deselected
            chosenColors.remove((String)colors[keywordId].
            getSelectedItem());
          }


          //then delete that pair from the tree table (Keywords and
      colors) since checkbox is deselected
      selectedKeywordsAndColors.remove((String)checkBoxes[
      keywordId].getText());

      //iterate through all the remaining entries of the hash map
      to print them in the text area
      Set entries = selectedKeywordsAndColors.entrySet();
      Iterator iter = entries.iterator();
      int number = 0;
      while(iter.hasNext())
      {
        number++;
        Map.Entry entry = (Map.Entry)iter.next();
        Object key = entry.getKey();
        Object value = entry.getValue();

        //add the chosen keyword in the chosen color in the text
        area below the tabbed pane
        textArea.append("\n" + number + ". " + key);
        textArea.append("\t" + value);
      }
      textArea.append("\n");
      break;
    }
  }
}
//check box is selected
else
{
 //increment the count (the number) of selected keywords up to now
  totalSelectedKeywords++;

  //keyword is selected by the user
  for (keywordId = 0; keywordId < KwNumber; keywordId++)
  {
    Object source = e.getItemSelectable();

    //find the keyword id
    if (source == checkBoxes[keywordId])
    {
      //if the maximum keywords have been already selected
      if(selectedKeywordsAndColors.size() ==
      SelectedKeywordNumber)

        //show a dialog box that no more than max number of
        keywords can be selected
        JOptionPane.showMessageDialog( null,
        "You can only select up to " + SelectedKeywordNumber +
        " keywords!" );
        checkBoxes[keywordId].setSelected(false);
        //decrement thecount the number of selected keywords up to
        now
        totalSelectedKeywords--;

      //if the user has not chosen a color for the previous
      checked keyword yet
      else if(totalSelectedKeywords > (chosenColors.size() + 1))

        //show a dialog box that a color must be selected before
        selecting the next keyword
```

```
            JOptionPane.showMessageDialog( null,
            "Please select a color for the previous keyword first!" );
            //decrements the count the number of selected keywords up
            to now
            checkBoxes[keywordId].setSelected(false);
          }
          //everything is going all right up to now
          else
          {
            //remember the keyword id to check its color
            selKwId = keywordId;
            textOfKeyword = (String) checkBoxes[selKwId].getText();
            colors[selKwId].setEnabled(true);

            //add the keyword with no color in the tree map; will be
            replaced when the color is chosen
            selectedKeywordsAndColors.put(textOfKeyword, "None");

            //append it to the text area
            textArea.append(checkBoxes[selKwId].getText());
            break;
          }
        }
      }
    }
  }

/** This class is an inner class  which is covenient when writing
event-driven programs; it adds or removes a color fromm the current
selection
and the text box; warns the user if color has been already used
*/
public class ColorAction implements ActionListener
{
  /** Implements the actionPerformed interface to carry out the
  functions of the class*/
  public void actionPerformed (ActionEvent colorEvent)
  {
    //find the color chosen
    colorOfKeyword = (String)colors[selKwId].getSelectedItem();

    int colorFlag = 0;
    //iterate through all the entries of the hash color map
    Set cols = chosenColors.entrySet();
    Iterator it = cols.iterator();
    while(it.hasNext())
    {
      Map.Entry color = (Map.Entry)it.next();
      Object key = color.getKey();
      Object value = color.getValue();

      //if color has  been used before
      if(key == colorOfKeyword)
      {
        colorFlag = 1;
        //warn the user; show a dialog box to choose another color
        JOptionPane.showMessageDialog( null, "" + colorOfKeyword +
        " color was used used for keyword \"" + value +
        "\".\n Please choose another color.");
```

```
          break;
        }
      }

    //color has not been used up to now
    if (colorFlag == 0)
    {
      //put the color in the hash table (to keep track of the used
      colors)
      chosenColors.put(colorOfKeyword, textOfKeyword);

      //put the pair keyword-color into the hash table
      selectedKeywordsAndColors.put(textOfKeyword, colorOfKeyword);

      //repaint the text area each time before showing the new entries
      textArea.setText("Selected keywords and associated colors :" +
      "\n");

      //iterate through all the entries of the tree map
      Set entries = selectedKeywordsAndColors.entrySet();
      Iterator iter = entries.iterator();
      int number = 0;

      while(iter.hasNext())
      {
        number++;
        Map.Entry entry = (Map.Entry)iter.next();
        Object key = entry.getKey();
        Object value = entry.getValue();

        //add the chosen keyword in the chosen color in the text area
        below the tabbed pane
        textArea.append("\n" + number + ". " + key);
        textArea.append("\t" + value);
      }
      textArea.append("\n");
      colors[selKwId].setEnabled(false);
    }
  }
}

/**Set the text for the label
 @param text  the text to which teh label will be set
 */
protected Component makeTextPanel(String text)
{
  JPanel panel = new JPanel(false);
  JLabel filler = new JLabel(text);
  filler.setHorizontalAlignment(JLabel.LEFT);
  panel.setLayout(new GridLayout(1, 1));
  panel.add(filler);
  return panel;
}

/** Add each component to the layout with the corresponding
constraints
@param c Component - the component to be added
@param constr - the constraints to the component
@param x specifies the column position in the upper left corner of the
component to be added
@param y specifies the row position in the upper left corner of the
component to be added
```

```
@param w specifies how many columns the component occupies
@param h specifies how many rows the component occupies
*/
public void add(Component c, GridBagConstraints constr, int x, int y,
int w, int h)
{
  constr.gridx = x;
  constr.gridy = y;
  constr.gridwidth = w;
  constr.gridheight = h;
  getContentPane().add(c, constr);

/** A class to establish the connection, perform the query and fill
the corresponding hash tables with the infor for the drawing
*/
public class Connect {
  /** the constructor for the class Connect
  Establishes the connection, performs the query and fills the
  corresponding hash tables with the infor for the drawing
  */
  public Connect()
  {
    Connection conn = null;
    PreparedStatement ps = null;
    ResultSet rs = null;

    try {
      conn = DBManager.getConnection();

      //hash map to hold column - color correspondence
      columnColorMapping = new HashMap();
      kwColor = new HashMap();

      //iterate through all the entries of the tree map
      selectedKeywordsAndColors
      Set allEntries = selectedKeywordsAndColors.entrySet();
      //System.err.println( "selected kws: " +
      selectedKeywordsAndColors.size() +  "\n" );
      Iterator iter = allEntries.iterator();
      //number of selected keywords
      int number = 0;
      //create an array of hashmaps
      geneInfoPerKeyword = new HashMap[selectedKeywordsAndColors.size
      ()];
      //iterate through the selectedKeywordsAndColors
      while(iter.hasNext())
      {
        Map.Entry entry = (Map.Entry)iter.next();
        Object key = entry.getKey();
        //make an array of all the keywords to be used for the
        annotation of the genes
        allSelectedKeywords[number] = (String) key;
        //System.err.println( "selected keyword#: " + number + " is "
        + key + "\n" );
        //the color of the keyword
        Object value = entry.getValue();
        //System.err.println( "it's color is : " +  value + "\n" );

        String currNum = "" + number;
```

```
//populate the hash table
columnColorMapping.put(currNum, value);
//System.out.println( "columnColorMapping# " + currNum +  " is
: " +columnColorMapping.get(currNum) + "\n" );
//key is the clolor and value is the keyword
kwColor.put(value, key);

String currentQuery =
    "select name, ac_kw.ac_num, kw " +
    "from ac_kw, kw_id, ac_name " +
    "where kw_id.kw_id = ac_kw.kw_id " +
    "and ac_name.ac_num = ac_kw.ac_num " +
    "and kw = ?";
ps = conn.prepareStatement(currentQuery);
ps.setString (1, allSelectedKeywords[number]);

//populate the array list with gene names
tree.writeOut(geneNames);
/*
for (int i = 0; i < (tree.getNumberOfGenes()); i ++)
{
  System.out.println((String)geneNames.get(i));
}
*/
rs = ps.executeQuery();
if ( rs != null )

  //for each keyword create a hash map to hold gene name and
  it's color
  geneInfoPerKeyword[number] = new HashMap();

  while ( rs.next() )
  {
    kw = rs.getString( "kw" );
    ac_num = rs.getString( "ac_num" );
    name = rs.getString( "name" );

    //populate the hash map for that query (kw) with gene name
    and the color for the keyword
    geneInfoPerKeyword[number].put(name, value);

    //iterate through the new entries to see them for
    debugging purposes
    /*
    Set entries = geneInfoPerKeyword[number].entrySet();
    Iterator iterations = entries.iterator();
    int num = 0;
    while(iterations.hasNext())
    {
      Map.Entry hashEntry = (Map.Entry)iterations.next();
      Object hashKey = hashEntry.getKey();
      Object hashValue = hashEntry.getValue();
      num++;
      //System.err.println("\n" + num + ". " + "Name: "+
      hashKey + "\t" + "Color: " + hashValue);
    }
    */
  }

else
```

```
    System.err.println("Result set is null");
   number++;
  }
 }
 catch ( DBManagerException e )
 {
  e.printStackTrace();
 }
 catch ( SQLException e )
 {
  e.printStackTrace();
 }
 finally
 {
  DBManager.close( rs, ps );
 }
}//end constructor

/**Method to get the number of keywords selected by the user */
public int getNumberOfKeywords()
{
 return selectedKeywordsAndColors.size();
}

/**Method to loop throught the HashMaps and the geneNames array and
to fill a keyword-color Array
2D array; raw =gene name; column = keyword; value = color*/
public void fillColorArray()
{
 //for each gene in the current tree
 for (int geneTreeEntry = 0; geneTreeEntry < tree.getNumberOfGenes
 (); geneTreeEntry++)
 {
  //create a new array to hold colors per gene
  currColorArray = new int[selectedKeywordsAndColors.size()] ;

  //System.out.println("Current Gene in tree: " + geneTreeEntry +
  " is " + geneNames.get(geneTreeEntry)+ " Total number of
  keywords is: "+selectedKeywordsAndColors.size());
  //for each selected keyword
  for ( int kws = 0; kws < selectedKeywordsAndColors.size(); kws++
  )
  {
   if (geneInfoPerKeyword[kws].containsKey((String)geneNames.get(
   geneTreeEntry) ))
   {
   //fill the keyword-color array
   currColorArray[kws] = 1;
   //colorMapping[geneTreeEntry][kws] = 1;
  }//leave it al zero
  else currColorArray[kws] = 0;
  /*
  if (kws ==  (selectedKeywordsAndColors.size() - 1))
  {
  System.out.print("The values for gene " +
  geneNames.get(geneTreeEntry)  + " are: ");
  for (int k = 0; k < selectedKeywordsAndColors.size(); k++)
  {
    System.out.print(currColorArray[k]);
   }
   System.out.println("");
   }
   */
   }//end inner for (kws)
   geneColorMapping.put(geneNames.get(geneTreeEntry),
   currColorArray);
   //System.out.println((String)geneNames.get(geneTreeEntry));
   /*
   //System.out.println("" + (int
   [])geneColorMapping.get((String)geneNames.get(geneTreeEntry)));
   int [] copy = new int [selectedKeywordsAndColors.size()];
   copy = (int
   [])geneColorMapping.get((String)geneNames.get(geneTreeEntry));
   System.out.print("The values of the copy array are: ");
   for (int k = 0; k < selectedKeywordsAndColors.size(); k++)
   {
    System.out.print(copy[k]);
   }
   System.out.println("");
   */
  }//end outer for loop geneTreeEntries
  //System.out.println(geneColorMapping);
 }//end fillColorArray()

 /**print geneColorMapping array elements for debugging purposes*/
 public void printColorMappingElements(HashMap geneColorMapping)
 {
  /*
  for (int i = 0; i < genes; i++)
  {
   for (int j = 0; j < 10; j++)
   {
    System.out.print(colorMapping[i][j]);
   }
   System.out.println("\n");
  }
  */
  //System.out.println(geneColorMapping);
  Set entries = geneColorMapping.entrySet();
  Iterator iter = entries.iterator();
  while (iter.hasNext())
  {
   Map.Entry entry = (Map.Entry)iter.next();
   Object key = entry.getKey();
   Object value = entry.getValue();
   System.out.println("key " + key + "value " + value);
  }
 }

 /**Return the specified colorMapping array element*/
 public int getCurrenrColorMappingElement(int curGene, int curKw)
 {
  return colorMapping[curGene][curKw];
 }
}//end class Connect()

/**The number of panels for each letter representing keywords starting
with that letter*/
```

```
public final int panelNumber = 24;
/**The height of the frame*/
public final int HEIGHT = 600;
/**The width of the frame*/
public final int WIDTH = 400;
/**The height of the text area*/
public final int HEIGHTTEXTAREA = 220;
/**The width of the text area*/
public final int WIDTHTEXTAREA = 350;
/**The height of the tabbed pane*/
public final int HEIGHTTABBEDPANE = 280;
/**The width of the tabbed pane*/
public final int WIDTHTABBEDPANE = 350;

//private ArrayList KwNumber;
private int KwNumber = 334;
private ArrayList allLetters;
private int MAXLETTERS = 26;
private int MAXKEYWORDS = 400;
private int letterKws = 0;
private JScrollPane scrollPane;
private JButton annotateButton;
private JButton cancelButton;
private JButton closeButton;
private int keywordId = 0;
private JLabel label;
private int i = 0;
private int kwSelectedCounter = 0;
private JCheckBox [] checkBoxes;
private JPanel [] panels;
private  Component [] tabPanels;
private char [] letters;
private JTabbedPane tabbedPane;
private JComboBox [] colors;
private CheckBoxListener keywordListener;
private ActionListener colorListener;
private GridBagConstraints constraints;
private int col1 = 0;
private int col2 = 1;
private int width = 1;
private int height = 1;
private int row = 0;
private String colorOfKeyword;
private String textOfKeyword;
private TreeMap selectedKeywordsAndColors;
private HashMap chosenColors;
private HashMap yeastKeywords;
private JTextArea textArea;
private JPanel buttonPanel;
private JPanel keywordsPanel;
private JPanel middlePanel;
private JPanel lowerPanel;
private JPanel upperPanel;
private int selKwId;
private int totalSelectedKeywords = 0;
private String ac_num;
private String kw;
private String  sec_ac;
private String  name;
private int kw_id;
```

```
  public final int SelectedKeywordNumber = 10;
  //array of hash tables
  private HashMap [] geneInfoPerKeyword;
  private  Connect annotateSession;
  private int []  currColorArray;
}//end class KeywordsFrameOptimized()
...

  /**The scrollable panel in which selected portion of tree is zoomed*/
  class ZoomPanel extends JPanel implements Scrollable

      /**Paint the component by drawing the zoomed portion of tree on
      it*/
      public void paintComponent(Graphics g)

              super.paintComponent(g);
              Rectangle rect=spr.getViewport().getViewRect();
              if (tree!=null)
              {
               tree.drawBig(g, (int)(rect.getY()/TreeOptions.sbv),
                           (int)((rect.getY()+rect.getHeight())/
                           TreeOptions.sbv+1), geneColorMapping,
                           columnColorMapping, kwColor);
                g.setColor(Color.black);
                //display the legend
                g.setFont(new Font (fontType, Font.BOLD,
                fontLegendSize));
                g.drawString("Legend:", moveLegendleft,TreeOptions.sbv);
                //display the icons next to keywords
                g.setFont(new Font (fontType, Font.BOLD,
                fontKeywordSize));
                incr = inc;
                width = add;

                for (col = 0; col < columnColorMapping.size(); col++)
                {
                    //find its color
                    String color = ((String)columnColorMapping.get(""+
                    col)).toLowerCase();
                    //System.out.print("Color is : " + color);
                    if (color.equals("red"))
                    {
                      g.setColor(Color.red);
                      drawRowLegend(g);
                    }
                    else if (color.equals("green"))
                    {
                      g.setColor(Color.green);
                      drawRowLegend(g);
                    }
                    else if (color.equals("orange"))
                    {
                      g.setColor(Color.orange);
                      drawRowLegend(g);
                    }
                    else if (color.equals("white"))
                    {
                      g.setColor(Color.white);
                      drawRowLegend(g);
                    }
                    else if (color.equals("yellow"))
                    {
                      g.setColor(Color.yellow);
                      drawRowLegend(g);
                    }
                    else if (color.equals("cyan"))
                    {
                      g.setColor(Color.cyan);
                      drawRowLegend(g);
                    }
                    else if (color.equals("black"))
                    {
                      g.setColor(Color.black);
                      drawRowLegend(g);
                    }
                    else if (color.equals("magenta"))
                    {
                      g.setColor(Color.magenta);
                      drawRowLegend(g);
                    }
                    else if (color.equals("pink"))
                    {
                      g.setColor(Color.pink);
                      drawRowLegend(g);
                    }
                    else if (color.equals("blue"))
                    {
                      g.setColor(Color.blue);
                      drawRowLegend(g);
                    }
                    incr = incr + add;
                    width = width + add;
                }
            }
        }

    /** Draw each row of the legend */
    public void drawRowLegend(Graphics g)
    {
      String currKw = (String)kwColor.get((String)columnColorMapping.get
      (""+ col));
      g.fillRect(moveLegendleft, TreeOptions.sbv+incr, TreeOptions.sbh-
      moveHor, TreeOptions.sbv-moveVert);
      g.setFont(new Font (fontType, Font.BOLD, fontSize));
      g.setColor(Color.black);
      g.drawString(currKw, moveKeywordleft,TreeOptions.sbv+width);
    }
    /**Create the panel*/
    ZoomPanel()
    {
            super();
            setBackground(TreeOptions.cback);
    }
  }

...
  /**
  Draw the zoomed tree, used in saving tree images. This way the entire
  ActionListener does not need to be overriden, only this function.
  @param g Graphics to draw on
```

```
@param from the starting height of drawing
@param to the ending height of drawing
@param colorMapping HashMap holds keyword-color correspondence
@param columnColorMapping HashMap holds correspondence between gene
and its keywords corresponding to the columns
*/

void drawZoomedTreeImage(Graphics g,int from, int to, HashMap
geneColorMapping, HashMap columnColorMapping)
{
        tree.drawBig(g,from,to, geneColorMapping, columnColorMapping,
        kwColor);
}
...

/**
  Draw the whole zoomed tree
  @param g Graphics to draw on
  @param from the starting height of drawing
  @param to the ending height of drawing
  @param colorMapping HashMap holds keyword-color correspondence
  @param columnColorMapping HashMap holds correspondence between gene
  and its keywords corresponding to the columns
*/
void drawWholeZoomedTreeImage(Graphics g,int from, int to, HashMap
colorMapping, HashMap columnColorMapping)
{
          tree.drawBigWhole(g,from,to, geneColorMapping,
          columnColorMapping, kwColor);
}
...
/**Load the tree, used so the constructor can be reused in the
inherited classes by only overriding this function.*/
void loadTree(File file)throws FileNotFoundException,ParsingException
{
          tree=new GeneTree(file);

          //get the number of genes
          genes = tree.getNumberOfGenes();

          //create the hash map to hold gebne names as keys and arrays
          with color info as values
          geneColorMapping = new HashMap();

          //get the number of genes
          //genes = tree.getNumberOfGenes();
          //create an array to hold the keyword-color code for each
          gene
          colorMapping = new int [genes][MaxKeywordNumber];
          //initialize all elements to zero to begin with
          for (int i = 0; i < genes; i++)
          {
            for (int j = 0; j < MaxKeywordNumber; j++)
            {
              colorMapping[i][j]= 0;
            }
          }
  }

  //print the values of the colorMapping array
```

```
//void printColorMapping(int [][] colorMapping)
void printColorMapping(HashMap geneColorMapping)
{
  /*
  for (int i = 0; i < genes; i++)
  {
    for (int j = 0; j < 10; j++)
    {
      System.out.print(colorMapping[i][j]);
    }
    System.out.println("\n");
  }
  */
  System.out.print(geneColorMapping);
}

...

//data members of MainFrame class
String fontType = "Serif";
int fontLegendSize = 14;
int fontKeywordSize = 10;
int fontSize = 12;
private int moveHor = 3;
private int moveVert = 8;
private int moveLegendleft = 280;
private int moveKeywordleft = 300;
private ArrayList geneNames;
private int [][] colorMapping;
private  final int MaxKeywordNumber = 10;
private String [] allSelectedKeywords;
private int genes;
private HashMap geneColorMapping;
private HashMap columnColorMapping;
private HashMap kwColor;
private int incr, inc = 9;
private int width = 15;
private int col = 0;
private int add = 15;
private int allYeastGenes = 6890;

...

JMenu kwAnnotationsmenu= new JMenu("Annotate");
JMenuItem annotate = new JMenuItem("Select Keywords");

...

public MainFrame()
{
    ...
    //array list to hold the gene names of the current input file;
    set the size at run time
    geneNames = new ArrayList();
    geneNames.ensureCapacity(allYeastGenes);
    allSelectedKeywords = new String[MaxKeywordNumber];
    ...
    //annotate genes
    annotate.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e){
      if (tree!=null)
      {
           KeywordsFrameOptimized keywordsFrame = new
           KeywordsFrameOptimized();
      }
      else
      JOptionPane.showMessageDialog(up,
      "There is no tree to annotate ", "No tree", JOptionPane.
      INFORMATION_MESSAGE);
      }
 });

//build the keywords frame with the help of JBuilder
annotationoptions.addActionListener(new ActionListener()
{
      public void actionPerformed(ActionEvent e)
      {
            //must open the optimized frame
            KeywordsFrameOptimized optionsFrame = new
            KeywordsFrameOptimized();
            //this part may be unnecessary
            //Validate frames that have preset sizes
            //Pack frames that have useful preferred size info, e.g.
            from their layout
            if (packFrame)
            {
              optionsFrame.pack();
            }
            else
            {
              optionsFrame.validate();
            }
            //Center the window
            Dimension screenSize = Toolkit.getDefaultToolkit().
            getScreenSize();
            Dimension frameSize = optionsFrame.getSize();
            if (frameSize.height > screenSize.height)
            {
              frameSize.height = screenSize.height;
            }
            if (frameSize.width > screenSize.width)
            {
              frameSize.width = screenSize.width;
            }
            optionsFrame.setLocation((screenSize.width - frameSize.
            width) / 2, (screenSize.height - frameSize.height) / 2);
            optionsFrame.setVisible(true);
            //the part above may be unnecessary
      }
 });
...
kwAnnotationsmenu.setToolTipText(
"Annotates genes with keyword information");
...
kwAnnotationsmenu.add(annotate);

...
}
```

```
    /**construct the menu*/
    void setUpMenu()
    {
        ...
        menubar.add(kwAnnotationsmenu);
        ...
    }

    /**enable/disable all parts of menu that are disabled if no tree is
    loaded*/
    void setEnabledMenu(boolean enabler)
    {
          annotate.setEnabled(enabler);
    }
}//end MainFrame class


==================================================================
package combinedtrial;

import java.io.*;
import java.util.*;
import java.sql.*;

public class DBManager
{
    private static Connection conn = null;
    private static String synchMember =
    "I'm a shared resource. Lock on me.";

    public static Connection getConnection () throws DBManagerException
    {
        if ( conn != null )
        {
            return conn;
        }

        // here is when we can get racing conditions - synchronize
        synchronized ( synchMember )
        {
            if ( conn == null )

                conn = createConnection();

            return conn;
        }
    }


    public static void releaseConnection( Connection c )
    {
        // do nothing for now - could be used later for connection
        pooling
    }


    private static Connection createConnection () throws
    DBManagerException
    {
        Connection conn = null;
        try {
            // load the DB driver's class dynamically
            Class.forName( "org.gjt.mm.mysql.Driver" );
            conn = DriverManager.getConnection(
            "jdbc:mysql://localhost/yeast_kw_db", "", "");

        // convert the ClassNotFoundException into a DBManagerException
        } catch ( ClassNotFoundException e ) {
            throw new DBManagerException(
            "DBManager:Couldn't load the DB driver!", e );

        // convert the SQLException into a DBManagerException
        } catch ( SQLException e ) {
            throw new DBManagerException(
            "DBManager:Couldn't connect to the database!", e );
        }

        return conn;
    }

    public static void close(
        ResultSet rs,
        PreparedStatement ps )
    {
        try {
            if ( rs != null ) {
                rs.close();
            }
        } catch ( SQLException e1 ) {
            // do nothing
        }
        try {
            if ( ps != null ) {
                ps.close();
            }
        } catch (SQLException e2) {
            // do nothing
        }
    }

    public static void close(
        PreparedStatement ps,
        ResultSet rs )
    {
        close( rs, ps );
    }
};

========================================================================
package combinedtrial;

/** This exception is intended to be returned by the DBManager class. It
should
 simplify the way clients call methods of the DBManager class.
*/
public class DBManagerException extends Exception
{
    private Exception rootCause;
    /**
     * This constructor helps to create the exception by passing not
     just some error
     * message at the logical level of the program, but also passing an
     exception
     * one might have caught. Later, if one wants to see the message of
     the
     * DBManagerException, one will get the exception he passed in here
     appended
     * after a logical error message.
     */
    public DBManagerException( String message, Exception e )
    {
        super( message );
        rootCause = e;
    }

    /**
     * This constructor simply passes the message along to parent class,
     since we
     * weren't given a root cause - another exception that one might
     save.
     */
    public DBManagerException( String message )
    {
        super( message );
        rootCause = null;
    }

    public String toString()
    {
        return getMessage();
    }

    public String getMessage()
    {
        if ( rootCause == null )
        {
            return super.getMessage();
        }
        else
        {
            return super.getMessage() + ": (" + rootCause.getMessage() +
            ")";
        }
    }
};

=================================================================
package combinedtrial;

import java.awt.*;
import java.io.*;
import java.math.*;
import java.util.*;

public class GeneTree
{
```

```
/**abstract class, from which everything (nodes and genes) in the
gene tree is inherited*/
abstract class TreeItem
{

        ...

        abstract void writeOut(ArrayList geneNames);
        abstract void drawBig(Graphics g, int wide, int from, int
        to, HashMap geneColorMapping, HashMap columnColorMapping,
        HashMap kwColor);
}//class TreeItem

/**inside tree node*/
class InsideTreeItem extends TreeItem
{

    ...
    //save all the gene names from the current file in an array
    void writeOut(ArrayList geneNames)
    {
            left.writeOut(geneNames);
            //System.out.println("Node "+gid+" sim: "+similarity);
            right.writeOut(geneNames);
    }


    //draw zoomed part of tree
    void drawBig(Graphics g, int wide, int from, int to, HashMap
    geneColorMapping, HashMap columnColorMapping, HashMap kwColor)
    {
      if (left.width>=from)
        left.drawBig(g,wide,from,to, geneColorMapping,
        columnColorMapping, kwColor);
      if (left.width<to)
        right.drawBig(g,wide+left.width,from-left.width,to-left.
        width, geneColorMapping, columnColorMapping, kwColor);
    }
}//class InsideTreeItem

/**Leaves of the tree, representing genes and containing gene
data.*/
class DataItem extends TreeItem
{

      ...

      //write to standard output, used for debugging purposes
      void writeOut( ArrayList geneNames)
      {
            //System.out.print(" name: "+name +" desc:
            "+description+" "+"\n");
            total = total + 1;
            //System.out.println(total);
            geneNames.add(new String (name));
            //System.out.println("Gene# " + total +" "+ geneNames);
      }


      //draw zoomed genes
      void drawBig(Graphics g, int wide, int from, int to, HashMap
      geneColorMapping, HashMap columnColorMapping, HashMap kwColor)
      {
        g.setFont(TreeOptions.font);
```

```
for (int i = 0; i < dataLength; i++)

  g.setColor(color[i]);
  g.fillRect(i*TreeOptions.sbh,wide*TreeOptions.sbv,
  TreeOptions.sbh-TreeOptions.sborder,TreeOptions.sbv-
  TreeOptions.sborder);
  g.setColor(TreeOptions.ctext);
  g.drawString(description,dataLength*TreeOptions.sbh+1,(
  wide+1)*TreeOptions.sbv-1);

//System.out.println("Gene name is " + name);

//if that gene name is contained in the geneColorMapping
hash map
if (geneColorMapping.containsKey(name))

  //must access the value of the hash that holds array with
  color mappings
  // create a temp array to hold the mappings
  int [] copy = new int [columnColorMapping.size()];
  copy = (int [])geneColorMapping.get(name);
  //System.out.print("The values of the new copy array are:
  ");
  //for (int k = 0; k < columnColorMapping.size(); k++)
  //{
    //System.out.print(copy[k]);
  //}
  //System.out.println("");
 //go through the values of the copy array and display the
 color icon if value is 1
  for (int col = 0; col < columnColorMapping.size(); col++)

    //if the value is 1, i.e. if the corresponding keyword
    annotates that gene
    if (copy[col] == 1)
    {
        //find its color
        String color = ((String)columnColorMapping.get(""+
        col)).toLowerCase();
        //System.out.print("Color is : " + color);
        if (color.equals("red"))
        {
          g.setColor(Color.red);
          g.fillRect((col + shiftCol)*TreeOptions.sbh,wide*
          TreeOptions.sbv +
                     shiftVert,TreeOptions.sbh - shiftHor,
                     TreeOptions.sbv - shiftVert);
        }
        else if (color.equals("green"))
        {
          g.setColor(Color.green);
          g.fillRect((col + shiftCol)*TreeOptions.sbh,wide*
          TreeOptions.sbv +
                     shiftVert,TreeOptions.sbh - shiftHor,
                     TreeOptions.sbv - shiftVert);
        }
        else if (color.equals("orange"))
        {
          g.setColor(Color.orange);
          g.fillRect((col + shiftCol)*TreeOptions.sbh,wide*
  TreeOptions.sbv +
             shiftVert,TreeOptions.sbh - shiftHor,
             TreeOptions.sbv - shiftVert);
}
else if (color.equals("white"))
{
  g.setColor(Color.white);
  g.fillRect((col + shiftCol)*TreeOptions.sbh,wide*
  TreeOptions.sbv +
             shiftVert,TreeOptions.sbh - shiftHor,
             TreeOptions.sbv - shiftVert);
}
else if (color.equals("yellow"))
{
  g.setColor(Color.yellow);
  g.fillRect((col + shiftCol)*TreeOptions.sbh,wide*
  TreeOptions.sbv +
             shiftVert,TreeOptions.sbh - shiftHor,
             TreeOptions.sbv - shiftVert);
}
else if (color.equals("cyan"))
{
  g.setColor(Color.cyan);
  g.fillRect((col + shiftCol)*TreeOptions.sbh,wide*
  TreeOptions.sbv +
             shiftVert,TreeOptions.sbh - shiftHor,
             TreeOptions.sbv - shiftVert);
}
else if (color.equals("black"))
{
  g.setColor(Color.black);
  g.fillRect((col + shiftCol)*TreeOptions.sbh,wide*
  TreeOptions.sbv +
             shiftVert,TreeOptions.sbh - shiftHor,
             TreeOptions.sbv - shiftVert);
}
else if (color.equals("magenta"))
{
  g.setColor(Color.magenta);
  g.fillRect((col + shiftCol)*TreeOptions.sbh,wide*
  TreeOptions.sbv +
             shiftVert,TreeOptions.sbh - shiftHor,
             TreeOptions.sbv - shiftVert);
}
else if (color.equals("pink"))
{
  g.setColor(Color.pink);
  g.fillRect((col + shiftCol)*TreeOptions.sbh,wide*
  TreeOptions.sbv +
             shiftVert,TreeOptions.sbh - shiftHor,
             TreeOptions.sbv - shiftVert);
}
else if (color.equals("blue"))
{
  g.setColor(Color.blue);
  g.fillRect((col + shiftCol)*TreeOptions.sbh,wide*
  TreeOptions.sbv +
             shiftVert,TreeOptions.sbh - shiftHor,
             TreeOptions.sbv - shiftVert);
}
```

```
                //display it in the color of the background
                else

                  g.setColor(Color.lightGray);
                  g.fillRect((col + shiftCol)*TreeOptions.sbh,wide*
                  TreeOptions.sbv +
                             shiftVert,TreeOptions.sbh - shiftHor,
                             TreeOptions.sbv - shiftVert);



      }//class DataItem
      ...

      private int total = 0;
      private int currGeneNum = 0;
      private int shiftCol = 13;
      private int shiftVert = 8;
      private int shiftHor = 3;
   }//GeneTree
```

APPENDIX B

PERL SCRIPT AND SUPPORTING FILES

This appendix contains the source Perl script used to parse the original SWISS-PROT file and to generate the files used to populate the local database (B.1). An excerpt of the output file KwACTableFile.txt is included for illustration purposes (B.2). The FillTables.bat file used to automate the population of the database is shown (B.3). Included also is the complete list of 333 keywords used in the yeast annotation process that was extracted from the SWISS-PROT original file (B.4).

B.1 Perl Script

```
# This program parses the yeast information from the Swiss-Prot
generated file which contains the primary
# and secondaryaccount information, description, keyword and gene names
of all yeast (Saccharomyces cerevisiae)
# genes and fills the files that are used to populate
# the tables in the yeast annotation database
# The statements used for debugging purposes are commented and left in
the program

use strict;

# The filename of the file containing the yeast data
my $yeastfilename = 'YeastAcDeGnKw.txt';

# the files to be created by the program
my $keywordIdFile = 'KwIdTableFile.txt';
my $ACdescrFile = 'ACDETableFile.txt';
my $keywordACfile = 'KwACTableFile.txt';
my $ACGNfile = 'ACGNTableFile.txt';
my $secACfile = 'SecACTableFile.txt';


my %kw_ids = ();
my @unique_keyword = ();
my @sortedUnique_keyword = ();
my @keywords =();
my @all_keywords =();
my %kw_seen = ();
my @allACs = ();
my %kw_appeared = ();
my %kw_index = ();
my $uniq_kw_counter = 1;
my $redund_kw_counter = 1;
my %ac_kw = ();

# First we have to "open" the file, and in case the
# open fails, print an error message and exit the program.
open( OUTkeywordIdFile, ">$keywordIdFile" ) or die
"Can't open output file <$keywordIdFile>: $!\n";
open( OUTACdescrFile, ">$ACdescrFile" ) or die
"Can't open output file <$ACdescrFile>: $!\n";
open( OUTkeywordACfile, ">$keywordACfile" ) or die
"Can't open output file <$keywordACfile>: $!\n";
open( OUTACGNfile, ">$ACGNfile" ) or die
"Can't open output file <$ACGNfile>: $!\n";
open( OUTsecACfile, ">$secACfile" ) or die
"Can't open output file <$secACfile>: $!\n";


my $file;
&ReadFile( $yeastfilename, \$file );

# Each element of array genes contains the  data for a single gene
```

```
my @genes = split( /^AC\s+/m, $file ); #??m means applied for each line
of the file
#print @genes;

# go through each gene's string and parse out the AC, DE, KW and GN
foreach my $gene_string ( @genes )

   # extract the accession numbers line
   my ($acc_string) = split( /;\n/, $gene_string );
   #print "$acc_string\n";

   # parse out the accession number(s)
   $acc_string =~ s/;\s/ /;
   #print "$acc_string\n";

   #hold primary (and secondary) accession number in an array
   my @acc_numbers = split( /\s/, $acc_string );
   #print "@acc_numbers\n";

   #save all the primary accession numbers in an array
   push (@allACs, $acc_numbers[0]);

   #if the array contains two Ac numbers
   if ( @acc_numbers == 2)

      my $sec_ac_table =
      "insert into second_ac (sec_ac, acc_num) values
      (\"$acc_numbers[1]\", \"$acc_numbers[0]\");\n";

      #write the array into the output file
      print OUTsecACfile $sec_ac_table;


   # remove the accesion numbers line from the gene string
   $gene_string =~ s/.*\n//;

   # process next gene string if gene has no more data
   #next if ( $gene_string eq '' );
   next unless ( $gene_string );

   # read all the annotations for this gene in an array of lines
   my ( @annotations ) = split( /[.;]?\n/, $gene_string );

   # remove the last element if it's empty (it should be)
   pop @annotations unless ( defined $annotations[$#annotations] );

   # arrays to keep each gene's DE, GN, KW data
   my @descriptions = ();
   my @name_lines = ();
   my @keyword_lines = ();
   my @unique_name = ();
   my %name_seen = ();
   #my ($description) = undef;
   my $description = undef;
   my $name = undef;
   my $keyword = undef;

   # parse out the annotations (keyword lines, gene name lines,
   descriptions)
   foreach my $ann_line ( @annotations )
```

```
{
   # the annotation is a description
   if ( $ann_line =~ m/^DE\s+/ )
   {
      #remove all (") from the descriptions - bothers the SQL
      $ann_line  =~ s/\"/ /g;

      #print "$ann_line\n";
      $ann_line =~ m/^DE\s+(.*)/;
      # remember the description and save it in the array of
      descriptions
      #$1 or \1 remembers what the first paretheses () hold, e.g.
      (.*)
      push( @descriptions, $1 );
   }

   # the annotation is a gene name line
   if ( $ann_line =~ m/^GN\s+/ )
   {
      #remove "(" and ")" from gene names
      $ann_line  =~ s/[\(\)]|AND\s+|OR\s+//g;

      $ann_line =~ m/^GN\s+(.*)/;
      # remember the gene name line
      push( @name_lines, $1 );
      #print"@name_lines\n";
   }

   # the annotation is a keyword_lines string
   if ( $ann_line =~ m/^KW\s+(.*)/ )
   {
      # remember the keyword_lines string
      push( @keyword_lines, $1 );
   }
} # end of each annotation line

# merge the descriptions into one string (one description per gene)
$description = join( " ", @descriptions );

my $ac_descr_table =
"insert into ac_descr (acc_num, descr) values (\"$acc_numbers[0]\",
\"$description\");\n";
print OUTACdescrFile $ac_descr_table;

# parse out the gene names from the gene name lines
foreach my $name_line ( @name_lines )
{
   #put all the names per line into an array names
   my @names = split ( /\s/, $name_line);

   #first record only non-redundant names for that gene
   foreach my $name ( @names )
   {
      push ( @unique_name, $name) unless $name_seen{$name}++;
   }
   foreach my $uniq_name ( @unique_name )
   {
      #correlate the gene's primary AC with all the alternative gene
      names
      my $ac_name_table =
      "insert into ac_name (ac_num, name) values
      "insert into ac_name (ac_num, name) values
      (\"$acc_numbers[0]\", \"$uniq_name\");\n";
      print OUTACGNfile $ac_name_table;
      #print $ac_name_table;
   }
}

# parse out the keywords from the gene keyword lines
foreach my $keyword_line ( @keyword_lines )
{
   #put all the keywords per line into an array keywords
   (@keywords) = split ( /\;\s/g, $keyword_line);

   # check whether each keyword in the array
   foreach my $new_kw (@keywords)
   {
      my $redundFlag = 0;
      #if unique keywords hash is empty
      if (!(%kw_index))
      {
         $kw_index{$uniq_kw_counter} = $new_kw;

         #correlate keyword ids with acc_num in the table kw_ac
         my $ac_kw_table =
         "insert into ac_kw (ac_num, kw_id) values
         (\"$acc_numbers[0]\", \"$uniq_kw_counter\");\n";
         print OUTkeywordACfile $ac_kw_table;
         #print $ac_kw_table;

         $uniq_kw_counter++;
      }
      else                    #the kw_index hash is not empty
      {
         #check for each key
         foreach my $key (keys (%kw_index))
         {
            #whether the value is the same as the new keyword read
            if ($kw_index{$key} eq $new_kw)
            {
               #if so, put it in the hash of redundant keywords
               $kw_appeared{$redund_kw_counter} = $new_kw;

               #correlate keyword ids with acc_num in the table kw_ac
               #even if we've encountered the word before, it should
               be there for that ac_num
               my $ac_kw_table =
               "insert into ac_kw (ac_num, kw_id) values
               (\"$acc_numbers[0]\", \"$key\");\n";
               print OUTkeywordACfile $ac_kw_table;
               #print $ac_kw_table;

               $redund_kw_counter++;
               #indicate we've encountered a redundant keyword

               $redundFlag = 1;

               #go directly to the next read keyword and skip the
               rest of the loop
               last;
            }
            } #end foreach;last comes here

            #if the keyword was not redundant, i.e. it's unique
            if ($redundFlag != 1)
            {
               #fill in the hash kw_index with indeces and corresponding
               unique keywords
               $kw_index{$uniq_kw_counter} = $new_kw;

               #correlate keyword ids with acc_num in the table kw_ac
               my $ac_kw_table =
               "insert into ac_kw (ac_num, kw_id) values
               (\"$acc_numbers[0]\", \"$uniq_kw_counter\");\n";
               print OUTkeywordACfile $ac_kw_table;
               #print $ac_kw_table;

               $uniq_kw_counter++;
            }
         }
      }#end each keyword
      push (@all_keywords, @keywords);
   }#end keyword line
   #print "@all_keywords";
} # each gene's string

#print all the unique keywords and their indeces
#and sort alphabetically the keywords
foreach my $key( sort {$kw_index{$a} cmp $kw_index{$b} } keys %kw_index
)
{
   #show key and its value
   #print "$key\t$kw_index{$key}\n";

   #correlate keyword ids with acc_num in the table kw_ac
   my $kw_index_table =
   "insert into kw_id (kw_id, kw) values (\"$key\",
   \"$kw_index{$key}\");\n";
   print OUTkeywordIdFile $kw_index_table;
   #print $kw_index_table;
}

#print "Redundant keywords: \n";
#print all the redundant keywords and their indeces
foreach my $k( keys (%kw_appeared) )
{
   #show key and its value
   #print "$k    $kw_appeared{$k}\n";
}

#print all the primary accession numbers
foreach my $ac ( @allACs )
{
   #print "$ac\n";
}

# save only non-redundant names for the genes
foreach my $keyword ( @all_keywords )
{
   push ( @unique_keyword, $keyword) unless $kw_seen{$keyword}++;
}
```

```
#sort all the unique keywords alphabetically
@sortedUnique_keyword = sort @unique_keyword;
my $kw_counter = 1;

foreach my $uniq_kw ( @sortedUnique_keyword )
{
   #fill the hash table with unique keywords for each numeric key
   $kw_ids{$kw_counter} = $uniq_kw;

   #print "$kw_counter\t$kw_ids{$kw_counter}\n";

   #correlate keyword ids with unique keywords in the table kw_id
   my $kw_id_table =
   "insert into kw_id (kw_id, kw) values (\"$kw_counter\",
   \"$kw_ids{$kw_counter}\");\n";
   #print $kw_id_table;

   $kw_counter++;
}

# Close the files
close OUTkeywordIdFile;
close OUTACdescrFile;
close OUTkeywordACfile;
close OUTACGNfile;
close OUTsecACfile;

# subroutine
sub ReadFile
{
   #function parameters
   my $file = shift @_;    #same as: my $file = shift;
   my $r_content = shift;

   local $/ = undef;      # ignore new line chars (must be "local" var)
   open( IN, "<$file" ) || die "Can't open $file: $!";
   $$r_content = <IN>;    #dereference it with $
   close( IN );
}
```

B.2 Excerpt of a Perl Script Output File

```
insert into ac_kw (ac_num, kw_id) values ("P31383", "1");
insert into ac_kw (ac_num, kw_id) values ("P31383", "2");
insert into ac_kw (ac_num, kw_id) values ("Q00362", "3");
insert into ac_kw (ac_num, kw_id) values ("P47096", "4");
insert into ac_kw (ac_num, kw_id) values ("P47096", "5");
insert into ac_kw (ac_num, kw_id) values ("P47096", "6");
insert into ac_kw (ac_num, kw_id) values ("P40433", "7");
insert into ac_kw (ac_num, kw_id) values ("P40433", "8");
insert into ac_kw (ac_num, kw_id) values ("P40433", "9");
insert into ac_kw (ac_num, kw_id) values ("P40433", "10");
insert into ac_kw (ac_num, kw_id) values ("Q12471", "7");
insert into ac_kw (ac_num, kw_id) values ("Q12471", "8");
insert into ac_kw (ac_num, kw_id) values ("Q12471", "9");
insert into ac_kw (ac_num, kw_id) values ("P38720", "4");
insert into ac_kw (ac_num, kw_id) values ("P38720", "11");
insert into ac_kw (ac_num, kw_id) values ("P38720", "12");
insert into ac_kw (ac_num, kw_id) values ("P53319", "4");
insert into ac_kw (ac_num, kw_id) values ("P53319", "11");
insert into ac_kw (ac_num, kw_id) values ("P53319", "12");
insert into ac_kw (ac_num, kw_id) values ("P25612", "4");
insert into ac_kw (ac_num, kw_id) values ("Q07747", "4");
insert into ac_kw (ac_num, kw_id) values ("P43547", "4");
insert into ac_kw (ac_num, kw_id) values ("P47182", "4");
insert into ac_kw (ac_num, kw_id) values ("P42884", "4");
insert into ac_kw (ac_num, kw_id) values ("P37898", "13");
insert into ac_kw (ac_num, kw_id) values ("P37898", "14");
insert into ac_kw (ac_num, kw_id) values ("P37898", "15");
insert into ac_kw (ac_num, kw_id) values ("P37898", "16");
insert into ac_kw (ac_num, kw_id) values ("P32357", "17");
insert into ac_kw (ac_num, kw_id) values ("P32357", "18");
insert into ac_kw (ac_num, kw_id) values ("P23542", "7");
insert into ac_kw (ac_num, kw_id) values ("P23542", "19");
insert into ac_kw (ac_num, kw_id) values ("P23542", "20");
insert into ac_kw (ac_num, kw_id) values ("P23542", "21");
insert into ac_kw (ac_num, kw_id) values ("P23542", "22");
insert into ac_kw (ac_num, kw_id) values ("Q01802", "7");
insert into ac_kw (ac_num, kw_id) values ("Q01802", "19");
insert into ac_kw (ac_num, kw_id) values ("Q01802", "20");
insert into ac_kw (ac_num, kw_id) values ("Q01802", "23");
insert into ac_kw (ac_num, kw_id) values ("Q01802", "24");
insert into ac_kw (ac_num, kw_id) values ("P27697", "23");
insert into ac_kw (ac_num, kw_id) values ("P27697", "24");
insert into ac_kw (ac_num, kw_id) values ("Q02486", "23");
insert into ac_kw (ac_num, kw_id) values ("Q02486", "25");
insert into ac_kw (ac_num, kw_id) values ("Q02486", "26");
```

B.3 The FillTables.bat File

```
mysqld
mysql yeast_kw_db < ACDETableFile.txt
mysql yeast_kw_db < ACGNTableFile.txt
mysql yeast_kw_db < KwACTableFile.txt
mysql yeast_kw_db < KwIdTableFile.txt
mysql yeast_kw_db < SecACTableFile.txt
mysqladmin shutdown
```

B.4 Extracted Keywords List

3D-structure
3Fe-4S
4Fe-4S
ANK repeat
ATP synthesis
ATP-binding
Acetylation
Actin-binding
Activator
Acyltransferase
Alkylation
Allosteric enzyme
Alternative initiation
Alternative splicing
Amino-acid biosynthesis
Amino-acid transport
Aminoacyl-tRNA synthetase
Aminopeptidase
Aminotransferase
Anion exchange
Antibiotic resistance
Antioxidant
Antiport
Antiviral
Arginine biosynthesis
Arginine metabolism
Aromatic amino acid biosynthesis
Arsenical resistance
Asparagine biosynthesis
Aspartyl protease
Autocatalytic cleavage
Autophagy
Biotin
Biotin biosynthesis
Branched-chain amino acid biosynthesis
Bromodomain
CBS domain
CF(0)
CF(1)
Cadmium
Cadmium resistance
Calcium
Calcium channel
Calcium transport
Calcium-binding
Calmodulin-binding
Capping protein

Carbohydrate metabolism
Carboxypeptidase
Cell adhesion
Cell cycle
Cell division
Cell shape
Cell wall
Centromere
Chaperone
Chitin degradation
Chitin-binding
Cholesterol biosynthesis
Chromatin regulator
Chromophore
Chromosomal protein
Cleavage on pair of basic residues
Coat protein
Coated pits
Cobalt
Coiled coil
Conjugation
Copper
Copper transport
Cyclin
Cycloheximide resistance
Cyclosporin
Cysteine biosynthesis
Cytoskeleton
DNA damage
DNA integration
DNA recombination
DNA repair
DNA replication
DNA replication inhibitor
DNA-binding
DNA-directed DNA polymerase
DNA-directed RNA polymerase
Decarboxylase
Dioxygenase
Dipeptidase
Dynein
Electron transport
Elongation factor
Endocytosis
Endonuclease
Endoplasmic reticulum
Exonuclease
Exosome
FAD
FMN
Fatty acid biosynthesis

Fatty acid metabolism
Fertilization
Flavoprotein
Folate biosynthesis
Formylation
Fusion protein
G-protein coupled receptor
GMP biosynthesis
GPI-anchor
GPI-anchor biosynthesis
GTP-binding
GTPase activation
Galactose metabolism
Gluconeogenesis
Glucose metabolism
Glutamate biosynthesis
Glutamine amidotransferase
Glutathione biosynthesis
Glycerol metabolism
Glycogen biosynthesis
Glycogen metabolism
Glycolysis
Glycoprotein
Glycosidase
Glycosyltransferase
Glyoxylate bypass
Glyoxysome
Golgi stack
Growth arrest
Growth regulation
Guanine-nucleotide releasing factor
Heat shock
Helicase
Heme
Heme biosynthesis
Histidine biosynthesis
Homeobox
Hydrogen ion transport
Hydrogen peroxide
Hydrolase
Hypothetical protein
Hypusine
Hypusine biosynthesis
Initiation factor
Inner membrane
Inositol biosynthesis
Intron homing
Ion transport
Ionic channel
Iron
Iron transport

Iron-sulfur
Isoleucine biosynthesis
Isomerase
Isoprene biosynthesis
Karyogamy
Keratin
Kinase
LIM domain
Leader peptide
Leucine biosynthesis
Leucine-rich repeat
Leukotriene biosynthesis
Ligase
Lipid degradation
Lipid transport
Lipid-binding
Lipoprotein
Lipoyl
Lyase
Lysine biosynthesis
Magnesium
Maltose metabolism
Manganese
Meiosis
Membrane
Metal-binding
Metal-thiolate cluster
Metalloprotease
Methionine biosynthesis
Methylation
Methyltransferase
Microsome
Microtubules
Mitochondrion
Mitosis
Monooxygenase
Motor protein
Multifunctional enzyme
Multigene family
Myosin
Myristate
NAD
NADP
Nitrate assimilation
Nonsense-mediated mRNA decay
Nuclear protein
Nuclease
Nucleosome core
Nucleotide biosynthesis
Nucleotide metabolism
Nucleotide-binding

Nucleotidyltransferase
One-carbon metabolism
Outer membrane
Oxidative phosphorylation
Oxidoreductase
Oxygen transport
Palmitate
Pantothenate biosynthesis
Pentose shunt
Peptide transport
Peroxidase
Peroxisome
Phenylalanine biosynthesis
Pheromone
Pheromone response
Phorbol-ester binding
Phosphate transport
Phospholipid biosynthesis
Phosphopantetheine
Phosphorylation
Plasmid
Polyamine biosynthesis
Polymorphism
Polyprotein
Polysaccharide degradation
Porin
Porphyrin biosynthesis
Potassium transport
Prenylation
Prenyltransferase
Primosome
Proline biosynthesis
Proline metabolism
Protease
Protease inhibitor
Proteasome
Protein biosynthesis
Protein kinase inhibitor
Protein phosphatase inhibitor
Protein splicing
Protein transport
Purine biosynthesis
Purine metabolism
Purine salvage
Pyridine nucleotide biosynthesis
Pyridoxal phosphate
Pyridoxine biosynthesis
Pyrimidine biosynthesis
Pyruvate
RNA-binding
RNA-directed DNA polymerase

RNA-directed RNA polymerase
Receptor
Redox-active center
Repeat
Repressor
Respiratory chain
Riboflavin biosynthesis
Ribonucleoprotein
Ribosomal frameshift
Ribosomal protein
Ribosome biogenesis
Rotamase
SH2 domain
SH3 domain
Sensory transduction
Septation
Serine biosynthesis
Serine esterase
Serine protease
Serine/threonine-protein kinase
Signal
Signal recognition particle
Signal transduction inhibitor
Signal-anchor
Sodium transport
Spermidine biosynthesis
Spliceosome
Sporulation
Steroid biosynthesis
Steroidogenesis
Sterol biosynthesis
Structural protein
Sugar transport
Symport
TPR repeat
Telomere
Tetrahydrobiopterin biosynthesis
Thiamine biosynthesis
Thiamine pyrophosphate
Thiol protease
Threonine biosynthesis
Topoisomerase
Toxin
Trans-acting factor
Transcription
Transcription regulation
Transducer
Transferase
Transit peptide
Translation regulation
Translocation

Transmembrane
Transport
Transposable element
Transposition
Tricarboxylic acid cycle
Tryptophan biosynthesis
Tyrosine biosynthesis
Tyrosine-protein kinase
Ubiquinone
Ubiquinone biosynthesis
Ubl conjugation
Ubl conjugation pathway
Unfolded protein response
Urea cycle
Voltage-gated channel
WD repeat
Xylose metabolism
Zinc
Zinc transport
Zinc-finger
Zymogen
cAMP
cAMP synthesis
cAMP-binding
mRNA capping
mRNA processing
mRNA splicing
mRNA transport
rRNA processing
rRNA-binding
tRNA processing

APPENDIX C

DATABASE TABLES

This appendix contains the database tables implemented in the MySQL database. They hold the information of the yeast gene keywords, description, primary and secondary account numbers and gene names.

The tables in MySQL database yeast_kw_db were created as follows:

1. ac_descr	(ac_num VARCHAR(6) NOT NULL,
	descr VARCHAR(240) NULL)
2. kw_id	(kw_id INT UNSIGNED NOT NULL,
	kw VARCHAR(60) NOT NULL)
3. second_ac	(sec_ac VARCHAR(6) NOT NULL,
	ac_num VARCHAR(6) NOT NULL)
4. ac_kw	(ac_num VARCHAR(6) NOT NULL,
	kw_id INT UNSIGNED NOT NULL)
5. ac_name	(name VARCHAR(80) NOT NULL,
	ac_num VARCHAR(6) NOT NULL)

REFERENCES

ArrayDB, Retrieved November 20, 2002 from the World Wide Web: http://www.niehs.nih.gov/Connections/2000/oct-nov/nov00-4a.htm.

ArrayExpress, Retrieved November 20, 2002 from the World Wide Web: http://www.ebi.ac.uk/microarray/ArrayExpress/arrayexpress.html.

Baxevanis, A. (2002). The molecular biology database collection: 2002 update. Nucleic Acids Research, 30, 1, 1-12.

BioKnowledge ™ Library, Retrieved November 20, 2002 from the World Wide Web: http://proteome.nih.gov:8000/may2001/garrels.html.

BioPerl Project Retrieved November 20, 2002 from the World Wide Web: (http://bio.perl.org/).

Brazma, A. & Vilo, J. (2000). Gene expression data analysis. FEBS Letters, 480, 17-24.

Brazma, A., Hingamp, P., Quackenbush, J., Sherlock, ., Spellman, P., Stoeckert, C., Aach, J., Ansorge, W., Ball, C. A., Causton, H. C., Gaasterland, T., Glenisson, P., Holstege, F. C., Kim, I. F., Markowitz, V., Matese, J. C., Parkinson, H., Robinson, A., Sarkans, U., Schulze-Kremer, S., Stewart, J., Taylor, R., Vilo J. & Vingron, M.(2001). Minimum information about a microarray experiment (MIAME) - toward standards for microarray data. Nature Genetics, 29, 365 - 371.

Brown, M. P. S., Grundy, W. N., Lin, D., Cristianini, N., Sugnet, C., Furey, T. S., Ares, M. Jr. & Haussler, D. (2000). Knowledge-based Analysis of Microarray Gene Expression Data By Using Support Vector Machines.
Retrieved November 20, 2002 from the World Wide Web:
http://citeseer.nj.nec.com/brown00knowledgebased.html.

Califano, A., Stolovitzky G. & Tu Y. (2000) Analysis of gene expression microarrays for phenotype classification. Submitted to the International Conference on Computational Molecular Biology
Retrieved November 20, 2002 from the World Wide Web:
http://citeseer.nj.nec.com/califano00analysis.html.

Celis, J. E., Kruhoffer, M., Gromova, I., Frederiksen, C., Ostergaard, M., Thykjaer, T., Gromov, P., Yu, J., Palsdottir, H., Magnusson, N. & Orntoft, T. F. (2000). Gene expression profiling: monitoring transcription and translation products using DNA microarrays and proteomics. FEBS Letters, 480, 2-16.

Cluster [Computer Software]. (1998), Retrieved November 20, 2002 from the World Wide Web: http://rana.lbl.gov/EisenSoftware.htm.

Dyck, T. (2002). Server Databases Clash. eWeek online magazine, http://www.eweek.com/print_article/0,3668,a=23115,00.asp.

DRAGON database, Retrieved November 20, 2002 from the World Wide Web: http://pc190-10.kennedykrieger.org/dragon.htm.

DuBois, P. (2000). <u>MySQL.</u> USA, New Riders Publishing.

Durbin, R., Eddy, S., Krogh, A. & Mitchinson, G. (1988). <u>Biological Sequence Analysis,</u> Cambridge, England, Cambridge University Press.

Dutilh, B., & Hogeweg, P. (1999). <u>Gene Networks from Microarray Data,</u> report Binf.1999.11.01, Bioinformatics, Utrecht University http://www-binf.bio.uu.nl/~dutilh/gene-networks/thesis.html.

Eisen, M., Spellman, Brown, P. O. & Botstein, D. (1998). <u>Cluster analysis and display of genome-wide expression patterns.</u> Proc. Natl. Acad. Sci., 95, 14863-14868.

ExpressDB, Retrieved November 20, 2002 from the World Wide Web: http://arep.med.harvard.edu/ExpressDB/ExpressDB.v102.help.htm.

Gene Expression Omnibus (GEO), Retrieved November 20, 2002 from the World Wide Web: http://www3.oup.co.uk/nar/database/summary/319.

Gene Ontology^{tm} Consortium, Retrieved November 20, 2002 from the World Wide Web: http://www.geneontology.org/#ontologies.

GeneCards project, Retrieved November 20, 2002 from the World Wide Web: http://bioinformatics.btk.utu.fi/genecards.

Gibas, C. &Jambeck, P. (2001). <u>Bioinformatics Computer Skills</u>. Sebastopol, CA, USA, O'Reilly & Associates., Inc.

Han, J. & Kamber, M. (2001). <u>Data Mining</u>. CA, USA, Morgan Kaufmann Publishers.

Hvidsten, T. R. , Komorowski, J., Sandvik, A. K., & Lægreid, A. (2001). <u>Predicting Gene Function from Gene Expressions and Ontologies, Pacific Symposium on Biocomputing,</u> 6, 299-310.

Jagota, A. (2001). <u>Microarray Data Analysis and Visualization.</u> Bioinformatics by the Bay Press, http://bioinformaticsbythebay.hypermart.net.

Kanehisa, M., Goto, S., Kawashima, S. & Nakaya, A. (2002). The KEGG databases at GenomeNet. Nucleic Acids Research, 30, 1, 42-46.

Kanehisa, M. (2000). Post-genome informatics. NY, USA, Oxford University Press.

Kavcic, M. & Zupan, B. (2001). FreeView. University of Ljubljana, Slovenia, and Baylor College of Medicine, Houston, TX, http://magix.fri.uni-lj.si.

Kyoto Encyclopedia of Genes and Genomes (KEGG), Retrieved November 20, 2002 from the World Wide Web: http://www.genome.ad.jp/kegg/).

Quackenbush, J. (2001). Computational analysis of microarray data. Nature Genetics, 2, 418-427.

Ramakrishnan, R. & Gehrke, J. (2000). Database Management Systems, USA, McGraw-Hill Companies, Inc.

Schwartz, R.L. & Christiansen, T. (1997). Learning Perl. Sebastopol, CA, O'Reilly & Associates, Inc.

Stanford Microarray Database (SMD), Retrieved November 20, 2002 from the World Wide Web: http://www.dnachip.org/.

SWISS-PROT database, Retrieved November 20, 2002 from the World Wide Web: http://us.expasy.org/sprot/.

TreeView [Computer Software]. (1998), Retrieved November 20, 2002 from the World Wide Web: http://rana.lbl.gov/EisenSoftware.htm.