5-2009

Preprocessing of microarray data and analysis and comparison techniques for the resulting graph structures

Jon A. Scharff
University of Tennessee

To the Graduate Council:

I am submitting herewith a thesis written by Jon A. Scharff entitled "Preprocessing of microarray data and analysis and comparison techniques for the resulting graph structures." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael A. Langston, Major Professor

We have read this thesis and recommend its acceptance:

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Jon A. Scharff entitled "Preprocessing of Microarray Data and Analysis and Comparison Techniques for the Resulting Graph Structures." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael A. Langston

Michael A. Langston, Major Professor

We have read this thesis
and recommend its acceptance:

Brynn H. Voy

Lynne E. Parker

Accepted for the Council:

Carolyn R. Hodges

Carolyn R. Hodges, Vice Provost and
Dean of the Graduate School

(Original signatures are on file with official student records.)

Preprocessing of Microarray Data and Analysis and Comparison Techniques for the Resulting Graph Structures

A Thesis
Presented for the
Masters of Computer Science
Degree
The University of Tennessee, Knoxville

Jon A. Scharff
May 2009

Dedication

This thesis is dedicated to my best friend, Tim. If it were not for faulty engineering, you would be far past me in your education and achievements.

Acknowledgments

I would like to thank my family. Without their support financially, emotionally, and otherwise, I would not likely have made it to this point in my education. This of course includes my parents, Jan and Jack Scharff, Jr., who have been accepting of any decision I have made from studying abroad, going on to graduate school, and taking time off to work in Japan, but it also includes my sister and my many friends that I have met along the way here.

I would also like to show appreciation to my teachers and professors in high school and beyond. Not only did they give me the time in class to help me improve myself, but they have given me many opportunities outside of the classroom that have shaped my education and fostered my desire to learn and create. Along this line, I would like to thank Dr. Michael Langston for working with me from my undergraduate education on the Oak Ridge National Labs (ORNL) summer grant and Dr. Brynn Voy both for working with me on that project and for offering me a graduate research assistantship before I had even applied for graduate school. Because of this opportunity, I was able to work towards my masters degree. I would like to thank both Dr. Langston and Dr. Voy again and thank Dr. Lynne Parker for taking the time to be on my thesis committee and having patience with me while I wrote this thesis.

I would like to acknowledge all the students and lab workers that I have worked with in Dr. Langston's lab, in classes, and at ORNL. Their support in research, study, and play contributed in no small part to my pushing forward.

Abstract

During our collaborations with scientists interested in high-throughput analysis of biological data, we have made much progress and facilitated some interesting findings using our clique-finding tools. However, we have also uncovered ways in which we can make our tools more efficient but have yet to write the programs to perform these tasks. Part of the problem is time constraints: in order to be useful to us, an application must be quite flexible and run efficiently. Programming such a tool is no small task, so we have resorted to scripting solutions that are geared to the specific task at hand. The first aim of this work is to produce a tool that is usable by both us and our collaborators to meet our common data processing needs.

Also during our collaborations, we have been tasked with finding new ways to help find potentially interesting data among a large amount of information that would be prohibitive to analyze by hand. One of our current tools is one that can take a graph and return all the maximal cliques, which, using real data, can be done in a reasonable amount of time. However, the list of maximal cliques itself is usually long and impractical to analyze by hand. Thus, we have needed to come up with new ways to sleuth out those genes and cliques that may be of most interest from a list of millions of cliques. The second aim of this work is to describe new methods that we have been using to achieve this.

Contents

Introduction **1**

1 Background **3**
1.1 Microarray Analysis . . . 3
1.2 Microarray Data Collection . . . 4
1.3 Clique Processing . . . 5

2 Data Processing **7**
2.1 File Formats . . . 7
2.2 Program Structure . . . 8
2.2.1 General Options . . . 9
2.2.2 Data Matrix Preprocessing Options . . . 9
2.2.3 Graph Options . . . 10
2.2.4 Analysis Outputs . . . 10
2.3 Memory Management . . . 11
2.4 File Format Routines . . . 12
2.5 Future Works . . . 12

3 Dataset Comparison **14**
3.1 Differential Correlation . . . 15
3.2 Differential Cliquification . . . 15
3.3 Future Works . . . 16

4 Clique Data Processing **17**
4.1 Clique Nuclei . . . 17
4.2 Future Works . . . 19

Conclusion **20**

Bibliography **21**

Vita **24**

List of Figures

1.1 Clustering types 4

2.1 Overall Program Structure 8
2.2 Preprocessing 9
2.3 Data Matrix to Graph 10
2.4 Graph Filtering 11

3.1 Differential Expression 14
3.2 Differential Correlation 16

4.1 Clique Nuclei 18
4.2 Preprocessing 19

Introduction

$\mathcal{NP}$-complete problems have long been approached with apprehension because they are widely believed to be intractable. In fact, intractability has often been used as an excuse either to stop work on solving a problem or to simplify or change it in some way to make the theoretical runtime bounds more practical. The desire to solve such problems exactly, however, has led to research into Fixed-Parameter Tractability (FPT), a field of study based on problems that have tractable solutions when a parameter of interest is bounded [10]. While the algorithms may still be exponential with respect to the parameter, they must be polynomial with respect to the general problem size. With such algorithms, the community has recently been able to solve a variety of $\mathcal{NP}$-complete problems that were once largely written off as unsolvable.

In our research group, we have been coding algorithms to solve real-world graph problems. One application we have been particularly successful in is clique [6]. By applying new algorithm techniques to solve the maximum clique problem, it has become possible to design practical algorithms using clique [1, 2, 11, 12]. In our research group, we have been successful in finding all maximal cliques in biological data, and we have been able to do so in graphs of tens of thousands of vertices and hundreds of thousands of edges in a reasonable amount of time [18, 19]. While there are other algorithms that do clustering [13], they tend to find highly connected components in a graph, of which clique is the "gold standard" [5].

Despite our ability to process such data in a reasonable amount of time, there are still many issues with the process from getting the data from our collaborators to returning results that they can use. In this work, I attempt to address some of these issues.

First, we had required from our collaborators a correlation matrix, showing the correlations of the expression of genes. However, this format is usually difficult to transfer for any reasonably sized dataset, so collaborations are hampered by the need to transfer large files, especially those that are done mainly at a distance. This problem is compounded when this exchange occurs multiple times, for instance because the original matrix is not optimal for our analysis, our collaborator has added new data, or they have determined some of the data is not usable.

Second, we had no uniform way of determining information about a graph once we receive it. As we continue working with various different researchers and with different data, we are able to determine what may be useful in a broad range of applications. Such functionality is best put into an application that can be used by the whole research team, as it allows for more uniform processing and results. It also saves us from coding the same functionality multiple times and provides a code base that can be made as efficient as possible so that we can learn what we want to know as soon as possible.

Third, we had received requests to compare two datasets, something that we were unsure of how to do. As this is a common request, coming up with such techniques would not only be helpful for our current collaborators but would likely be useful to future ones as well.

Finally, once we have obtained the maximal clique listings, they tended to be very long and almost as difficult to process as the original data itself. While we have experimented with some techniques to find useful information in these files, which I will discuss further in chapters 1 and 4, having more ways to do so would add value to our analyses.

Before discussing how we have worked toward solutions to these issues, I will first give some background on the type of work we have been doing. Then, I will discuss a program that I wrote to deal with preprocessing the data that we receive from our collaborators as well as to perform basic analyses on the resultant graphs. In the last two chapters, I will discuss new methods that we have begun using to deal with comparing two datasets as well as a different method of dealing with the overwhelming amount of data that tends to result from the enumeration of maximal cliques.

Chapter 1

Background

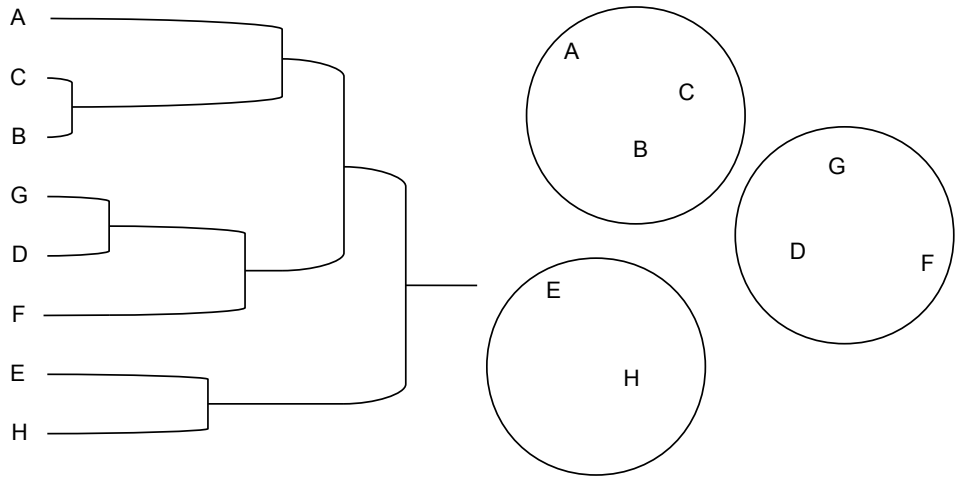
To discuss the techniques new to our process, I need to provide some background on it. First, I would like to give a brief overview of microarray analysis and data collection, as it gives insight into the problems we have encountered and our proposed solutions. Then, I would like to give a summary of our clique processing. As these explanations are meant only to provide a limited background for those unfamiliar with them, they are by no means complete. However, they should suffice for understanding the rest of this paper.

1.1 Microarray Analysis

First, it is necessary to discuss why we are able to use our clique finding tools on microarray data and find anything of use to our collaborators. Clustering has been done on microarray data before we began working on this problem, but we feel that our method is superior in many ways to what had previously existed [13]. Before going into the differences of clustering methods, however, I should briefly discuss the relevance of using clustering techniques on microarray data.

Microarrays give us relative expression values for many different genes at once. We can conduct many microarray experiments over different conditions, in which the cells are being affected by different types of stresses, and use the resulting expression values to calculate a correlation coefficient for each pair of genes. Then, we can cluster genes that are highly correlated with the hypothesis that genes that are co-expressed are more likely to share common regulatory mechanisms. In other words, co-expression implies co-regulation. While this is not necessarily the case, it is a common belief, and there is work on trying to determine how likely co-regulation is when co-expression is given [4]. We can also extend this idea to include genes that are highly negatively correlated. In such cases, there is a possibility that a promoter of one gene is a suppressor of another. Thus, correlation tells us likelihood of co-expression, which hints at co-regulation.

Due to the tens of thousands of genes that mammals have and the complexity of the gene networks that are involved in many genetic-level reactions that we would like to study, simply looking at high correlations is usually not enough to narrow down the search field for our collaborators. Because two or more networks being active together under different conditions is possible, a high correlation between a pair of genes in itself does not necessarily give us much information. However, if we are able to find a group of genes that are highly correlated, then we can use known information about some of the genes in the group to determine the likelihood that what we have found is not only likely to be a network of genes working together but also that the group of genes is likely to



(a) Hierarchical (b) Centriod (k=3)

Figure 1.1: Two major types of clustering methods

be due to the stimulus being studied in the current experiment [5]. This is where clustering comes in handy.

Most clustering algorithms fit into one of two categories, hierarchical and centroid. Hierarchical algorithms use a tree structure where the leaves are single genes and every other node is the union of the genes in its children nodes (fig. 1.1(a)). Thus the root node contains all of the genes. Centroid methods try to fit all the genes into a specified number of clusters (fig. 1.1(b)). Both processes lead to groupings that are disjoint or, in the case of hierarchical, do not contain overlapping sets that do not have a subset-superset relationship. Because of this, it is hard to describe a relationship between some genes, such as three genes, A, B, and C, where gene A is highly correlated with genes B and C, but genes B and C do not have a high correlation. Such a case could feasibly exist if gene A works in multiple gene networks. Clique does not require making all result sets disjoint. Furthermore, since the main goal of most clustering algorithms is to get groupings that are highly correlated, by using clique you can enforce that all pair-wise correlations in any result group is no smaller than the threshold you set.

Finding the maximum clique size in a graph is just the beginning, however. What we really want to find is the set of all maximal cliques, or those cliques that cannot have another vertex from the graph added and still be a clique. This enumeration takes exponential time, but knowing the maximum clique size allows us to estimate the run time of these algorithms on a particular graph more accurately. If it is decided that the maximum clique size is too large, it is easy enough to raise the threshold on the correlations to remove edges from the graph and decrease the maximum clique size. Thus, since we are able to find all maximal cliques within real biological datasets in a reasonable amount of time, there is no reason not to use clique in such an application.

1.2 Microarray Data Collection

This section is intended only to summarize the method for collecting the data that we have been using. It is not meant to be exhaustive but only to show where I chose to leave tasks to our collaborators and where I thought we could handle processing the data. It is also important to

understand some of the issues that occur in, though not necessarily that are unique to, microarray experiments.

Microarrays themselves are glass slides with spots that have been placed to attract certain genetic information from cellular material washed over the slide. While spots and genes are not necessarily a one-to-one correspondence, for the purposes of this paper, I will treat them as such. If this is not the case, we can either work with the spots instead, or, with our collaborator's help, we can convert the spot information to gene information. The genetic material that is washed over the slide has been marked with a dye so that its presence on the slide may be observed via a slide scanner. This scanner gives the relative intensities of the dye at each spot and thus the relative abundance for each gene in the wash that was detectable by the slide. To remove noise that is inherent in microarray experiments, normalization is done [14, 16]. This process is specific to the nature of microarrays and is usually included in software to process the scanned slides. As such, it is something that is best left to our collaborators. After normalization, what is left is a numeric value for each gene telling us about the relative abundance of the material produced from each gene: this is the data that we are interested in being able to process.

Many times, the microarray data that we work with was not originally intended for the type of analysis we do, thus we have to be able to work with data that is not necessarily ideal for our tools. For example, we have worked with data that was originally collected for use with differential expression, something I will discuss further in chapter 3. The data for such analysis tends to have many biological replicates and can be done with as few as two different conditions [3]. With the correlation analysis that we do, it is much more useful to have many conditions at the expense of replicated data. However, when we do have a dataset that does not have enough separate conditions but does have replicated data, we can sometimes treat the different replicates as different conditions if there is enough variation within the replicates.

In any case, the datasets that we are asked to work with usually comprise of tens of thousands of genes over tens of arrays. Without the capability to do correlations in house, we must receive a triangular matrix of gene-to-gene correlations; this tends to result in a data file that, even compressed, is in the hundreds of megabytes. Such a file is not convenient to transfer, especially when we are collaborating with people long distance. On the other hand, the original data from which the correlations are calculated is usually only in the tens of megabytes uncompressed. This is much easier to transfer over networks and store on disk for long periods of time. Working with the data in this state also opens up new possibilities for us. We are now able to experiment with different normalization techniques to give better correlation values, determine if we can combine biologically replicated data or if we must treat each replicate separately, try out different correlation coefficients, drop arrays from the correlations or add new arrays to them, run permutations or other statistical tests on the data, and other such analyses.

Therefore, I felt it would be beneficial for us to be able to work with data after normalization to deal with microarray-data-specific issues had been finished but before normalization to clean the data for a particular analysis had occurred. We should still be able to work with data that has been normalized for correlations or correlation data calculated by our collaborators, but being able to work with the data in a previous state would be advantageous.

1.3 Clique Processing

Our clique tool chain previously started with an input of a correlation matrix containing a correlation for the expression patterns of every pair of genes. We treat this matrix as a weighted

graph where the genes are vertices and the correlation between two genes is the weight of the edge between the corresponding vertices. We then determine a threshold value to convert the weighted graph to an unweighted one for use in our clique tools. The determination of this threshold is currently not formalized, and so we may sometimes try to learn information about the unweighted graph at different thresholds to decide what threshold to use at first. We usually start with a threshold that leaves us with a very sparse graph as this causes our clique codes to run much more quickly and gives much more manageable result sets. Next, we run our maximum clique codes on the unweighted graph to determine the upper bound on clique size. Because we are translating our clique problem to a vertex cover problem and using FPT techniques on the converted graph, we are able to solve this problem in a reasonable time on most graphs resulting from real data [2].

Once we know the maximum clique size in the current graph, we can talk to our collaborators to see if they want us to change the threshold so that we get an acceptable maximum size. Not only are we concerned with the run time of our maximal clique codes, but we also do not want to return cliques that are too large to be of use, as they will likely be processed by hand. Now we can run our maximal clique codes to find all maximal cliques in the graph. A maximal clique is one such that no vertex in the graph that is not a member of the clique is connected to all vertices in the clique, i.e. there is no clique in the graph that is a superset of the clique in question. These are the clusters that we then return to our collaborator. However, the number of resulting clusters is usually too great to be processed by hand. In the past, we have further processed this data in different ways. One way is to determine the number of maximal cliques each gene is a member of. The idea is that genes that are in many cliques may have an important function related to the experimental condition. We may also send paracliques, highly connected subgraphs found algorithmically using the original weighted correlation matrix and the maximal clique list, to our collaborator [7]. The way that paraclique is defined, the resulting clusters are disjoint, but we may also look at the connectivity between paracliques to see genes that appear to be involved in multiple clusters [8]. Finally, we have looked to other information available about genes in a given clique that might hint at their possible co-regulation [5].

Chapter 2

Data Processing

To address the issues of the initial transfer of data from our collaborators to our group and the analysis of that data, I wrote a program that would take gene expression data and generate a correlation matrix of the similarity in the expression patterns of pairs of genes. This matrix can then be used to find cliques using our existing tools as discussed in chapter 1. The program needed to be able to handle current data sizes of over 20,000 genes as well as be able to expand to 40,000 or more genes in the foreseeable future; as the correlation matrix is of size $O(v^2)$, memory footprint was an issue. I also wanted to make sure that many different similarity measurements were supported: while we have mainly used Pearson's correlation coefficient, there may be reason to use others. Third, I needed to make sure that many preprocessing tasks could be done in-house both to alleviate the burden on our collaborators and to allow us to test the robustness of our tools to different types of data that might not be of direct interest to our collaborators. In the same program, I wanted to include some of the common graph analysis techniques that we had found useful in dealing with biological datasets in the past. Next, I wanted my program to remain as portable as possible. To this end, I used only standard C libraries and tested new features as I added them on Sun Sparc, Linux Intel, and Windows Intel operating-system-processor combinations. While this required me writing more code myself than might have been necessary, it allows for my research group to use the program under various situations on campus or while at conventions. It also makes sharing the program with our collaborators and others much easier. Also, as this tool needed to be usable both by my team and our collaborators from the command line as well as in scripts, strict file, argument, and error checking was necessary. Finally, to make sure all our future programs would be able to read in the file formats that I wrote out, I had to make sure my graph reading and writing routines were usable in a wide variety of applications. This meant creating functions that were more flexible than what was required just for my program.

In the following sections, I will first introduce the file formats that I will refer to in the rest of this chapter. Then, I will discuss the structure of the program, including all the options available to the user. I will then talk about memory-management issues that I decided to address during the design process. Next, I will talk about the file-reading routines that I designed. Finally, I will discuss further changes that may make the program more useful and/or easier to use.

2.1 File Formats

There are three major file formats that we use with this program. One is a data-matrix, which we use to hold the raw microarray data that we get. We also have two different graph formats: a

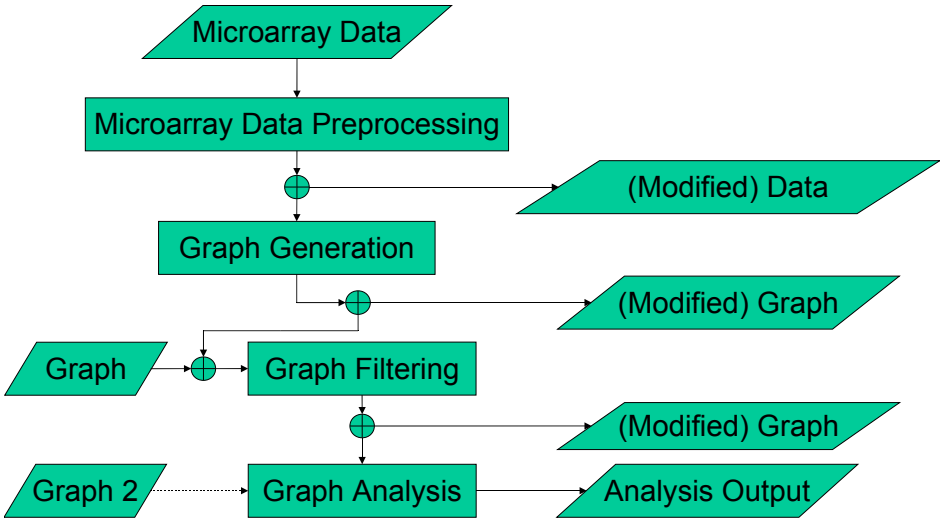


Figure 2.1: A flowchart of the basic program structure. More detailed descriptions of the parts follow in the subsections.

lower-triangular matrix or an edge list. All file formats are tab delimited, i.e. values in a row are separated with the tab character. A more detailed description of each of these formats follows.

The data-matrix file contains a header with the number of rows followed by either the number of columns or a name for each column. The column name need not be unique. Each row represents a variable, such as a gene, and each column represents a different condition, such as a microarray. A single row will start with the variable name followed by the values corresponding to that variable for each experiment. Either a numeric value or a blank value is valid.

A lower-triangular-matrix file has a header telling the number of rows and rows with the variable or vertex name followed by numeric or blank values representing the rows of the matrix where the upper triangle and main diagonal are omitted.

An edge list has a header telling the number of vertices and rows, and rows representing the edges with the names of the two variables or vertices involved in the edge and, optionally, the weight of the edge.

2.2 Program Structure

When writing the program, I wanted it to be completely argument driven for easy scripting and batch processing. To facilitate scripting even further, I wanted to allow for various types of input and output formats as well as any input to be read for standard input or a file. There are also numerous different operations that the user may want to perform on the data at different points in the process of converting the initial data matrix to a graph format. This led to a rather large amount of arguments and many possible paths of execution the details of which are illustrated in figure 2.1 and in the rest of this section.

The data to be manipulated may be either a matrix of data, referred to in the rest of the paper as a data matrix, or a graph. This data may be provided via standard input or in a file that is identified in the arguments. The output may be a data matrix, a graph, or one of many different

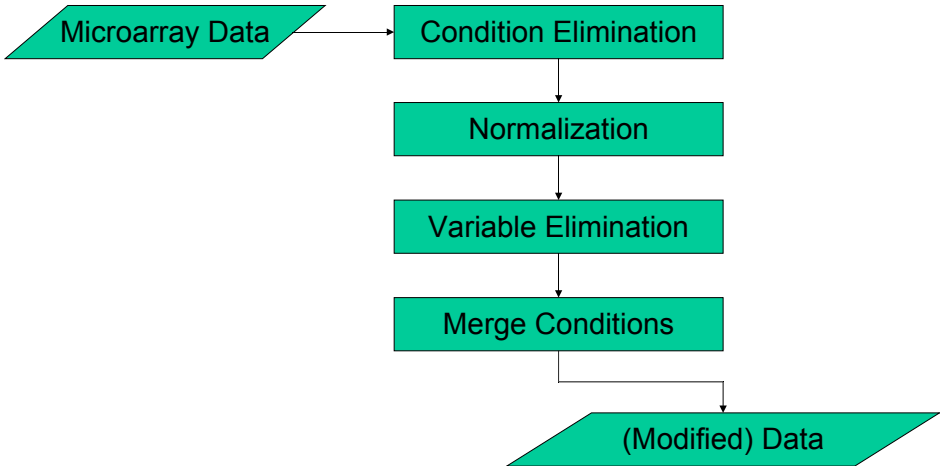


Figure 2.2: A flowchart of the preprocessing options. All preprocessing steps are optional. Thus, the resultant data may be the same as the input data.

analyses on the input data. This output may be print to standard out, or it may be put in a file identified in the arguments. Each of the options for processing the data is discussed in detail below.

2.2.1 General Options

There is a help option to display all possible arguments along with their usage, and a short description. There is also an option to print out a comment header to the output file that includes information about how the program was run. This is especially useful as it allows reproducibility and uniform processing of multiple datasets at different times. There are also options to read the input from a file instead of standard input and print to a file instead of standard output. Finally, there are options to limit variables or vertices to a provided list or exclude certain variables vertices. Both options allow for the list to be provided on the command line, as a file, or via standard input.

2.2.2 Data Matrix Preprocessing Options

When processing data from its data matrix form, there are many manipulations we may want to perform before generating the resultant graph (fig. 2.2). First, we may want to limit the conditions to use in generating the graph. We may do so by listing the names or column numbers of the conditions to omit in the command line, in a file, or via standard input. In the case of providing names, all columns with the name listed will be removed. Next, there is an option to normalize the data in the columns by subtracting the average and dividing by the standard deviation. The average may be calculated as either the mean or the median, and normalization is done before any variables are removed from the file, if such an option is specified. After normalization, the user has the option to merge data columns with the same name. This only applies if a name has been applied to each column in the data matrix header and is useful in biological applications as there is often redundancy built in to an experiment. These columns may be merged by taking the mean or median of all values with the same name, and the user may also specify the minimum number of conditions with the same name that must have numeric values (as opposed to blank values) to consider the merged value meaningful; otherwise it will be assigned a blank value.

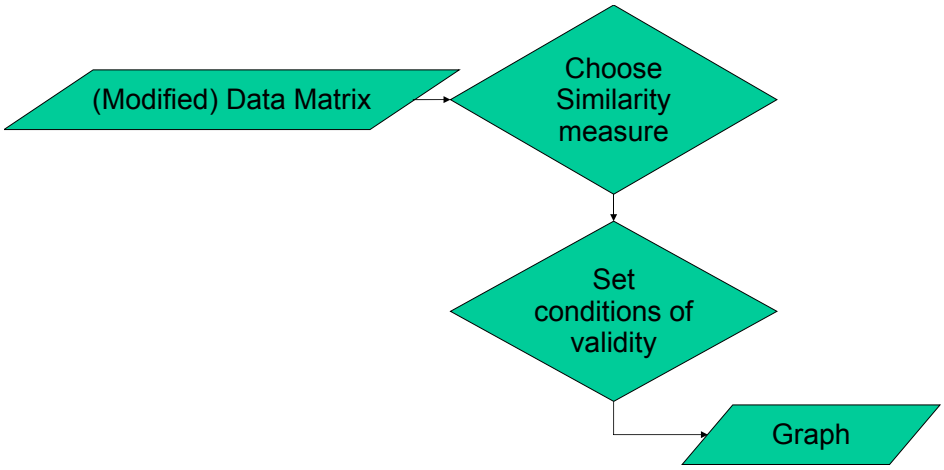


Figure 2.3: A flowchart of the options when converting the data to a graph.

2.2.3 Graph Options

When going from a data matrix to a graph, there are a few options that the user may use (fig. 2.3). First, the user may specify a similarity measure type. Although it is heavily modified, the basis of the code for many of these measurements as well as some other calculations such as mean and median comes from the University of Tokyo's C Clustering Library [9]. Next, there is an option to specify the minimum number of conditions having numeric values for both conditions for a similarity measurement to be considered valid; otherwise, it will be assigned a blank value. The user is also able to run the similarity measurement multiple times for each variable pair dropping a specified number of conditions and then determining the final measurement from the resulting values. The user may take the mean or median of the results as well as taking the maximum or minimum of the results.

After the graph has been generated, there are yet more options the user may specify (fig. 2.4). First, the user may specify a threshold value. All values less than the threshold specified will be left blank in the resulting graph. After applying a threshold to the graph, the user may limit the graph to only the neighborhood of certain variables as a list on the command line, in a file, or in the input to the program. Finally, the user may specify whether to output a weighted graph, and, if the output is weighted, what precision to limit the weights to, with a maximum precision of 4 digits past the decimal. If the user opts for the graph to be unweighted, then, in the lower-triangular matrix format, a 1 will specify an edge, a 0 will specify an edge below the threshold, and a blank value will specify a similarity measurement that was considered invalid.

2.2.4 Analysis Outputs

Aside from data matrix and graph outputs, the user can also have the results of various analyses be the output. The user may choose to print statistics about the resulting graph, such as number of vertices, number of edges, zero-valued edges, and negative-valued edges, and average, minimum, and maximum weights. The user may also print an edge-weight distribution to see how many edges have a certain edge weight when rounded to the specified precision. If the user wants to know information about connectivity, he or she may find out how many edges there are from each vertex to all other vertices or to a set of vertices provided by the user.

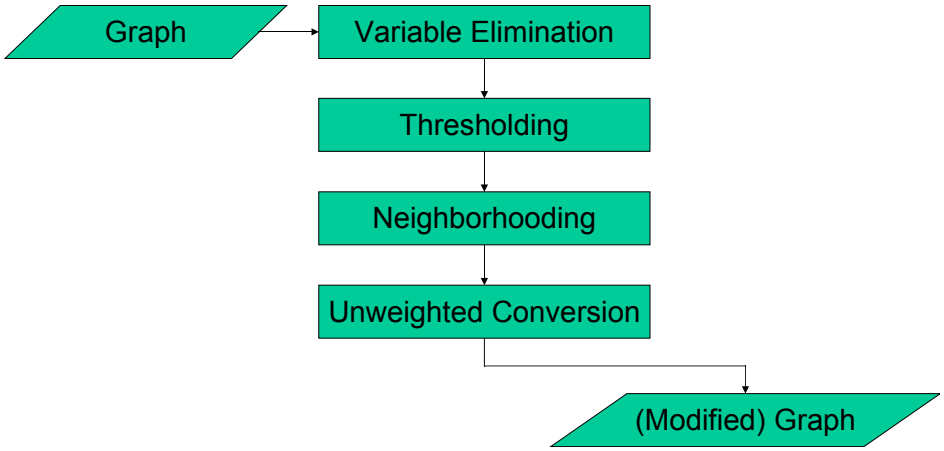


Figure 2.4: A flowchart of the options available to a user after the graph is generated. All are optional.

For comparing weights in two different graphs, the user has two choices. One is a two-dimensional distribution showing what weight an edge has in the first graph versus what weight the edge has in the second graph, and the other a list of statistics about the weights of the edges in a second graph grouped by their weight in the first graph. Finally, the user may obtain information about corresponding edges in two graphs that have significantly different weights based on an upper and lower threshold as chosen by the user. With all three of these comparisons, the second graph may be provided in a file identified in the command line or via standard input. However, the second graph currently must be in the lower-triangular-matrix file format described earlier.

2.3 Memory Management

One issue I wanted to avoid was a high memory requirement. This meant avoiding using doubles to store the resultant correlation matrix. I needed to allow for missing values in the correlation matrix due to the possibility of missing data values. Therefore, I only needed 202 possible values, as we are usually only interested in two places past the decimal in the range from -1 to 1. While I this could work with only one byte, I decided to allow for a little more precision by moving to a two-byte data value, allowing for four decimal places of precision. This cuts the required memory for the correlation matrix to half that of a matrix using single-precision floating-point values, or to just under 1.5GB for a 40,000-gene correlation matrix. As this was the maximum size in the foreseeable future for microarray data and it could fit into the memory size of many of the computers available for our use, I felt the two-byte size was convenient.

Once functionality for comparison of two graphs became a necessary feature, I made sure that only one graph was loaded into memory to keep the maximum footprint to a reasonable size. However, keeping efficiency in mind, I also wanted to be sure that reading the graph from file was done in a serial manner to avoid excessive random disk accesses. This meant I needed to be able to search the graph that is loaded into memory efficiently to find the edge corresponding to the one at the current location in the file. For this, I felt the best trade between time and space was with a red-black tree structure.

Aside from these choices, I was also careful to free all memory that I could as soon as it was no longer needed. I also made sure that any memory that was allocated inside a function was freed before the function exited either normally or on an error.

2.4 File Format Routines

This program uses a few different file formats, but they are all tab delimited. Thus, I chose to write a single tab-delimited-file reader. I wanted a reader that could skip blank lines and comments but one that did not automatically do so. I also needed a reader that would not skip blank fields, as these are used as value placeholders. With any good text-file reader, I had to worry about new lines. Since the definition of a new line is different for text files native to Windows, Unix/Linux, and Macintosh operating systems, I needed a reader that was able to handle all three types of new lines regardless of the current platform. Thus, I chose to code a reader myself. I use this reader for reading in all input files. I decided only to look for the three main types of new lines; however, coding for additional types of new lines would not be excessively difficult, especially those of one character in length. I chose to make the delimiter character a tab character, but I left it as a defined constant so that it could be changed at compile time if necessary. The reader should work for any delimiter of any length as specified as a compile-time constant. The comment delimiter may be set at run time. A comment is the leading white space characters before the comment delimiter, the delimiter itself, and all characters following the delimiter up until the next new line.

Since our research group works with many different graph algorithms, I felt that we might benefit from very general graph i/o routines. While making them general also makes them somewhat more difficult to use, they allow us to be more consistent in the graph format we use. All programs that use these functions should be able to share graph files, both edge lists and lower-triangular matrices, with each other. In order to make the functions useful for a wide range of applications, I included many options to allow the programmer some flexibility in how he or she stores the graph in memory. While the order of the values in memory is fixed, the programmer may specify such things as the number of bits that corresponds to each value and the value that corresponds to an invalid entry in the graph. Thus he or she may be storing a weighted graph that requires 32 bits per value, or he or she may be using a single bit to specify whether an edge exists or not.

2.5 Future Works

While the program is certainly capable in its current state, and it has been tested and used for an extended period of time without problem, there are still some actions that should be considered to improve its usability and efficiency. First, because of the very flexible nature of the program, there are many options that can be difficult to remember. As such, it may be beneficial to split the program into two or more parts such that each new program would be much easier to remember than the current whole. This would lead to more file i/o in general, but it may be an acceptable trade off. Second, there are other functions that are often used by my team members that I have not yet included in the functionality of the program. Including this functionality would simplify their work to some degree. Third, the documentation for how to use the program and of the code itself should be augmented to make use and maintenance easier. Finally, multithreading should be considered to improve efficiency of graph generation. While making the program multithreaded but

still run on multiple platforms is a difficult task, the fact that the newer generation of processors for home computers often support running threads simultaneously makes such an effort more practical.

Chapter 3

Dataset Comparison

cDNA microarrays are well suited to collect data from two datasets at the same time for comparison [15]. Conventionally, biologists have used differential expression (fig. 3.1), which is a Student's t-test performed on each gene in the two datasets. The idea is that, if the expression values after normalization for a gene in each dataset are different enough, then the difference in the two datasets is likely the cause of this difference in expression. However, this tells nothing about changes that occur in networks. Since correlation analysis attempts to find putative networks, we also want to be able to compare putative networks in two datasets to determine what changes might be caused by the experimental conditions.

One possibility is to look at the cliques that differentially expressed genes occur in and see what changes occur. However, this might not let us see network-wide changes that are more subtle than the t-test allows for or that are spread through multiple genes' expression patterns. Beyond this, we have come up with two other possibilities to determine changes that occur in how genes interact. First, we have extended the idea of differential expression, which is a gene-to-gene or vertex-to-vertex comparison, to what we have called differential correlation, which is an edge-to-edge comparison. Second, we have even further explored the differential idea with cliques. Instead of just looking at cliques containing differentially-expressed genes, we also choose genes that are

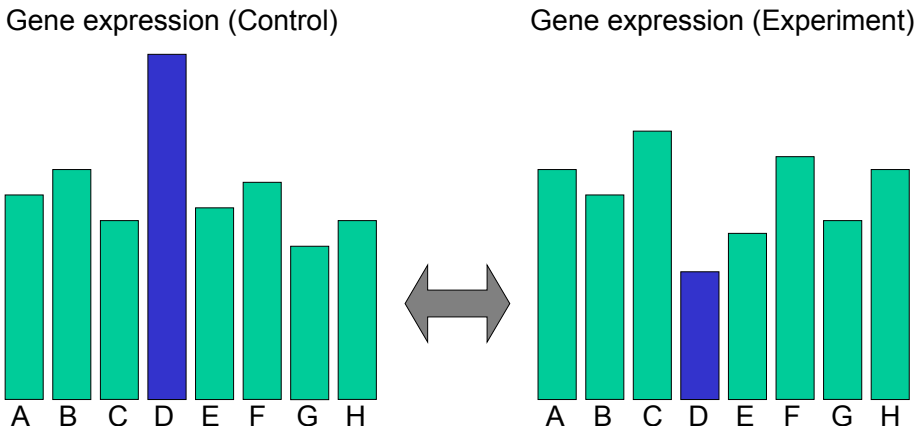


Figure 3.1: Differentially expressed genes are those such as D which have a very different expression relative to the other genes in the two different conditions being compared.

Algorithm 3.1 Differential Correlation

$x \leftarrow$ the correlation between genes g and h in dataset A
$y \leftarrow$ the correlation between genes g and h in dataset B
$upper \leftarrow$ the upper threshold
$lower \leftarrow$ the lower threshold
if $|x| \geq upper$ and ($|y| \leq lower$ or $\text{sign}(x) \neq \text{sign}(y)$) **then**
  g and h are differentially correlated
else if $|y| \geq upper$ and ($|x| \leq lower$ or $\text{sign}(x) \neq \text{sign}(y)$) **then**
  g and h are differentially correlated
else
  g and h are not differentially correlated
end if

involved in a large number of cliques in one dataset but not the other. Both methods appear to give interesting results.

3.1 Differential Correlation

Differential correlation came directly from the idea of differential expression. Where as differential expression looks for genes that have a significant difference in gene product under the experimental condition when compared to the control, differential correlation looks for pairs of genes with a significantly different correlation between the experimental condition and the control. For differential correlation, we choose two thresholds and find the gene pairs with correlation above the upper threshold in absolute value in one dataset and either below the lower threshold in absolute value or of opposite direction (+ vs. −) in the other dataset (alg. 3.1). This functionality was added as one graph analyses included in the program described in chapter 2.

When we do this with our example datasets, we find single genes involved in many of the gene pairs that are differentially correlated. When we create a graph using the differential correlation information, these genes appear in the center of star shapes (fig. 3.2). In order to determine if this star shape is significant or not, we have run permutation tests by swapping control and experimental labels. If the pattern of single genes being involved in many differential correlations is due to the relationship between the control and experimental data, then we hypothesize that, by mixing the data by swapping corresponding control and experimental array data, we should see these shapes disappear in the differential correlation graph. This, however, does not appear to be the case, as the star shapes appear even when we scramble the data. This may be a result of the relatively small amount of data that we have for this dataset, or it could be that the star shape itself may not be significant. However, even if the latter is the case, the genes involved in differential correlations may still be significant. We have been able to find some genes that are perhaps of interest from the list of differentially correlated genes [18], but these genes may just be involved by chance. We would like a way to determine the significance of the relationships that we find with this technique.

3.2 Differential Cliquification

Using our clique tools, we are able to find the maximal cliques in the resulting graphs of both the control and experimental microarray datasets. Therefore, we wanted to be able to use these lists to

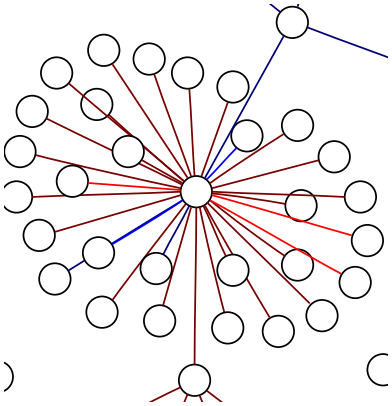


Figure 3.2: Blue edges represent genes highly correlated in dataset A but not dataset B, and red edges represent genes highly correlated in dataset B but not dataset A. Brighter edges represent correlations that are of opposite direction (+ vs. −) in the two datasets. For this example, the upper threshold is |.85| and the lower threshold is |.25|.

find differences between the two sets. Expanding on the idea of differential correlation and thinking about the star shapes, we began to wonder if the edges appearing in one graph but not the other might be edges that are involved in cliques. If this is the case, then we hypothesized that the genes in the center of the stars should either be involved in many cliques in one dataset but not the other or be involved in cliques with different genes in the two different datasets. While we did find that the genes in the center of the differential correlation stars were involved in differing amounts of cliques in the two datasets, they were not involved in a large amount of cliques, usually only having membership in less than one percent of the total number of cliques in either dataset.

This led us to look for genes that were in a large proportion of cliques in one dataset but not the other, a technique we have called differential cliquification or disproportionate abundance. This is yet another way to see a gene that appears to be working with many other genes under one condition but is not so strongly correlated to other genes in the other condition. Thus, we may hypothesize that the gene in question or the network(s) that the gene is involved in are somehow affected by the experimental condition. This technique also found genes and interactions that merit further research [18].

3.3 Future Works

While both of these methods have shown promise in application, they are still not yet formalized. Currently, we have only been able to guess what should be considered significant when determining whether a gene is differentially correlated or differentially cliquified. However, if we were to look into statistical tests to determine significance, not only could we have more confidence in the results we return, but we could also automate the process of identifying genes that are differentially correlated or differentially cliquified much more easily.

Chapter 4

Clique Data Processing

Upon finding all maximal cliques in a graph, we are still left with a lot of data that must be processed, as a graph might typically have hundreds of thousands to millions of maximal cliques. We have dealt with this issue in the past in a couple of ways. First, we have looked for genes that are in many different maximal cliques. While these genes may be generic ones that are involved in normal cell processes, they may also be highly involved with other genes due to the experimental conditions. In particular, if a gene is involved with other genes that are thought to be related to the experimental condition then that gene may be a good candidate for further study. We have had success with this type of analysis in the past [5]. Second, we have developed an algorithm named paraclique, which has also lead to exciting results [7, 17]. Third, we have used batch-processing tools such as Gene Ontology Tree Machine to find cliques that are enriched with genes that are known to be involved in certain processes, which allows us to determine if a clique is likely to contain genes of interest to the research at hand [5].

Since the initial question we are usually asking is whether there are groups of genes that are working together, a natural extension of finding single genes that occur in many cliques is finding subsets of genes that occur in many cliques. Finding such information about a group of genes gives more confidence that they are truly working together and thus further justifies verifying biologically whether the identified group is related to the experimental condition. However, while such a question is easy enough to ask, answering it is quite another matter. The first worry is how to decide upon the definition of "many." In particular, we want to be able to find genes working together often, but we do not necessarily want to ignore genes that are not in many cliques as these may still be of interest. Thus, we can modify our query so that, instead of using the word many, we ask about subsets of genes in which each gene in the subset does not appear in many cliques without the other genes in the subset. One method of answering this question is illustrated in the following section.

4.1 Clique Nuclei

Clique nuclei began as an attempt to narrow down the long lists of cliques that we had to deal with from any particular dataset. One thing we had noticed was that some genes appeared in many cliques. When we made a list of the cliques a gene was a member of, we also noticed that some of the same genes often appeared in cliques together. Seeing this, we began to wonder how we might find subsets of genes that are in many cliques together (fig. 4.1). Not only would this give us one more way to filter our clique list, but it might also give us more confidence that this subset of genes

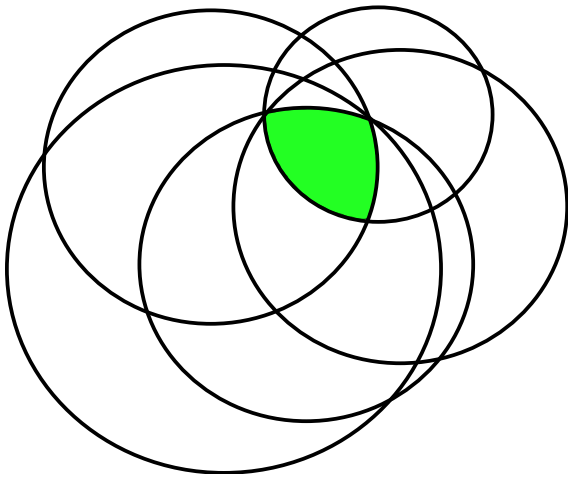


Figure 4.1: The green area represents the genes in the overlap of all the cliques (represented by circles). These are the genes we are interested in finding via clique nuclei.

is indeed related in some fashion. However, giving an absolute definition to how many cliques a subset of genes should be in to be considered significant is unrealistic. Since there are some genes involved in hundreds of thousands of cliques for one dataset and some genes that are involved in relatively few, it would be likely that a group of genes that always show up together but are only members of a few thousand cliques would get washed away by the genes involved in many more cliques.

This called for a much more flexible definition of significance. What we really wanted to find were groups of genes that, if they appeared in a clique, they almost always appeared together. To achieve this, I decided to take our clique lists and treat them as a new dataset. In other words, instead of microarrays being the data columns to create correlations between two genes, each clique became a data column with a value of 1 or 0 assigned to each gene if that gene was in the clique or not, respectively. For genes that could not be in the clique due to lack of data, the data point was considered invalid and left blank. Thus we have a bit vector for each gene that we can use to generate a similarity coefficient. As the data was binary in nature, I chose Jaccard similarity coefficient, a correlation coefficient designed for binary data. For any two genes, this coefficient results in the ratio of the number of cliques that both genes are in to the number of cliques that at least one is in. However, if either gene could not have been in a clique due to lack of data, that clique was not considered in the comparison, even if the other gene was in the clique.

This led to a new problem, however. Since our datasets usually contain hundreds of thousands to millions of cliques, I was dealing with a data matrix that could not fit into memory and might also be a problem storing in just one file on disk for some file systems. To deal with this, I split the file into chunks, and, since the data matrix would not work with the memory model of the program I had created earlier, I created a new program to generate a data matrix split over several files and then generate a graph file using Jaccard similarity coefficient on the data matrix.

As I did not need the flexibility of the program described in chapter 2, I kept the functionality very simple (fig. 4.2). For each gene, I scanned the maximal clique list file to determine what cliques the gene was a member of. If the gene appeared in the nth clique, I entered a one in the data matrix corresponding to the gene and clique. If the gene did not appear in the nth clique, I used the original graph file to determine if the gene was not connected to another gene in the clique due

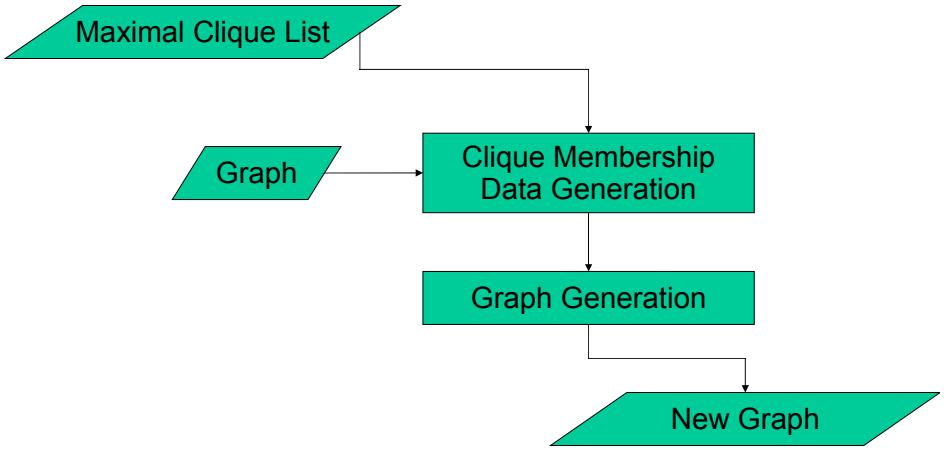


Figure 4.2: A flowchart of the preprocessing options. All preprocessing steps are optional. Thus, the resultant data may be the same as the input data.

to a lack of data. If this was the case, I entered a null value in the data matrix, otherwise I inserted a zero. Note that I assign a null value due to lack of data even if one of the other correlations might have been less than the threshold. This is because more data might cause these correlations to rise above the threshold. Once the data matrix is completed, a new graph is generated from it.

With the resulting graph file, we can then use our extablished clique tools to look for clique nuclei. While they are interesting in themselves, we may also further extend the idea of differential cliquification. Since we now have a group of genes that commonly occur together in one dataset, we can look to see how often they occur together in a second dataset. We might hypothesize that the genes in a clique nucleus in one dataset that are not highly correlated in a second dataset are genes affected by the differing conditions of the two datasets and are thus of interest. If the clique nucleus is highly correlated in both datasets, then we may begin to look at the differences in the cliques that the set of genes is involved in as well as the difference in expression of the genes in each dataset.

4.2 Future Works

This method is relatively new, and, as such, we are not sure where the results of such analysis might lead us. More work needs to be done in determining the significance of the results. If this methods lives up to its promise, then the code that was written to test the concept should be modified to make it more flexible. The current hard disk requirements to store the massive data matrix make it difficult to apply to our data. As such, a program that recomputes clique-membership vectors so that they need not be stored on disk, while slower, may be more practical for more general use.

Conclusion

In our collaborations, we have run against many problems, some of which I have attempted to solve in this work. We are now much more flexible in the data and people we can work with, as potential collaborators have to do less processing before giving us data, and the files we can receive can be much smaller. We also have more to offer in the way of dataset comparisons and parsing out potentially interesting information from the massive lists that our techniques tend to generate.

Throughout our work, we continue to face new challenges that require new solutions. While most of our current work has been dealing with finding putative gene networks, we are always interested in different applications and different angles to the problems we have been working on. Not only does this allow us to generalize our current techniques, but it also gives us a chance to create new ones that we may in turn use on our current datasets.

Bibliography

Bibliography

[1] F. N. Abu-Khzam, M. A. Langston, and W. H. Suters. Fast, effective vertex cover kernelization: A tale of two algorithms. In *Proceedings of Computer Systems and Applications, 2005*, Cairo, Egypt, 2005. The 3rd ACS/IEEE International Conference. doi: 10.1109/AICCSA.2005.1387015.

[2] Faisal N. Abu-Khzam, Michael A. Langston, Pushkar Shanbhag, and Christopher T. Symons. Scalable parallel algorithms for fpt problems. *Algorithmica*, 45(3):269–284, 2006. doi: 10.1007/s00453-006-1214-1.

[3] David B. Allison, Xiangqin Cui, Grier P. Page, and Mahyar Sabripour. Microarray data analysis: From disarray to consolidation and consensus. *Nature Reviews Genetics*, 7:55–65, January 2006. doi: 10.1038/nrg1749.

[4] Dominic J Allocco, Isaac S Kohane, and Atul J Butte. Quantifying the relationship between co-expression, co-regulation and gene function. *BMC Bioinformatics*, 5(18), 2004. doi: 10.1186/1471-2105-5-18.

[5] Nicole E. Baldwin, Elissa J. Chesler, Stefan Kirov, Michael A. Langston, Jay R. Snoddy, Robert W. Williams, and Bing Zhang. Computational, integrative, and comparative methods for the elucidation of genetic coexpression networks. *Journal of Biomedicine and Biotechnology*, 2005(2):172–180, 2005. doi: 10.1155/JBB.2005.172.

[6] I Bomze, M Budinich, P Pardalos, and M Pelillo. The maximum clique problem. In P. M. Pardalos D-Z Du, editor, *Handbook of Combinatorial Optimization*, pages 1–74. Kluwer Academic Publishers, 1999.

[7] E. J. Chesler, S. Shou L. Lu, Y. Qu, J. Gu, J. Wang, H. C. Hsu, J. D. Mountz, N. E. Baldwin, M. A. Langston, J. B. Hogenesch, D. W. Threadgill, K. F. Manly, and R. W. Williams. Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function. *Nature Genetics*, 37(3):233–242, 2005.

[8] E. J. Chesler and M. A. Langston. Combinatorial genetic regulatory network analysis tools for high throughput transcriptomic data. *Proceedings, RECOMB Satellite Workshop on Systems Biology and Regulatory Genomics*, 2005. San Diego.

[9] M. J. L. de Hoon, S. Imoto, J. Nolan, and S. Miyano. Open source clustering software. *Bioinformatics*, 20 (9):1453–1454, June 2004. doi: 10.1093/bioinformatics/bth078.

[10] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, New York, 1999.

[11] M. R. Fellows and M. A. Langston. On search, decision, and the efficiency of polynomial-time algorithms. *Journal of Computer and System Sciences*, 49(3):769–779, 1994.

[12] M. A. Langston. Practical fpt implementations and applications. In *Proceedings of Parameterized and Exact Computation: First International Workshop*, Bergen, Norway, 2004. IWPEC 2004. doi: 10.1007/b100584.

[13] John Quackenbush. Computational analysis of microarray data. *Nature Reviews Genetics*, 2(6):418–27, June 2001.

[14] John Quackenbush. Microarray data normalization and transformation. *Nature Genetics*, 32 Suppl:496–501, December 2002. doi: 10.1038/ng1032.

[15] Mark Schena, Dari Shalon, Ronald W. Davis, and Patrick O. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467–470, October 1995.

[16] Gordon K. Smyth and Terry Speed. Normalization of cdna microarray data. *Methods*, 31(4):265–73, December 2003. doi: 10.1016/S1046-2023(03)00155-5.

[17] D. L. Tabb, M. R. Thompson, G. K. Khalsa-Moyers, N. C. VerBerkmoes, and W. H. McDonald. Ms2grouper: group assessment and synthetic replacement of duplicate proteomic tandem mass spectra. *Journal of the American Society of Mass Spectrometry*, 16(8):1250–1261, 2005.

[18] Brynn H. Voy, Jon A. Scharff, Andy D. Perkins, Arnold M. Saxton, Bhavesh Borate, Elissa J. Chesler, Lisa K. Branstetter, and Michael A. Langston. Extracting gene networks for low-dose radiation using graph theoretical algorithms. *PLoS Computational Biology*, 2(7), 2006. doi: 10.1371/journal.pcbi.0020089.

[19] Y Zhang, F. N. Abu-Khzam, N. E. Baldwin, E. J. Chesler, M. A. Langston, and N. F. Samatova. Genome-scale computational approaches to memory-intensive applications in systems biology. In *Proceedings of the ACM/IEEE SC 2005 Conference*, 2005. doi: 10.1109/SC.2005.29.

Vita

Jon Allan Scharff was born in Memphis, Tennessee, on February 3, 1981, the son of Jack Gordon Scharff, Jr. and Jan Margaret Warwick Scharff. Jon graduated from Germantown High School in Germantown, Tennessee, and he continued on to The University of Tennessee, Knoxville. During his undergraduate career, Jon studied abroad for two semesters in Japan before returning to graduate in the fall of 2004 with Bachelor of Science in both computer science and math and a minor in Japanese. Jon continued to graduate school under the mentorship of Dr Michael Langston, and he received a graduate research assistantship with Dr. Brynn Voy of Oak Ridge National Labs. In the summer of 2005, Jon completed his assistantship and, lacking only his thesis, went to Japan to teach English. During his time abroad, Jon was able to make slow progress, and he returned in the fall of 2006 briefly to defend. Returning to Japan, he continued making revisions to his thesis, and he completed his Master of Computer Science, officially graduating in spring of 2009. Jon then returned to Japan to finish his work there and is currently contemplating what to do next.