University of Tennessee, Knoxville
TRACE: Tennessee Research and Creative Exchange

Doctoral Dissertations | Graduate School

8-2011

Transcriptomic Data Analysis Using Graph-Based Out-of-Core Methods

Gary L Rogers
grogers3@utk.edu

Follow this and additional works at: https://trace.tennessee.edu/utk_graddiss

 Part of the Theory and Algorithms Commons

Recommended Citation

Rogers, Gary L, "Transcriptomic Data Analysis Using Graph-Based Out-of-Core Methods. " PhD diss., University of Tennessee, 2011.
https://trace.tennessee.edu/utk_graddiss/1122

This Dissertation is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a dissertation written by Gary L Rogers entitled "Transcriptomic Data Analysis Using Graph-Based Out-of-Core Methods." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Michael A. Langston, Major Professor

We have read this dissertation and recommend its acceptance:

Michael W. Berry, Jian Huang, Arnold Saxton, Brynn H. Voy

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 dissertation written by Gary L. Rogers Jr. entitled "Transcriptomic Data Analysis Using Graph-Based Out-of-Core Methods." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Michael A. Langston
Major Professor

We have read this dissertation
and recommend its acceptance:

Michael W. Berry

Jian Huang

Arnold Saxton

Brynn H. Voy

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

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

Transcriptomic Data Analysis Using Graph-Based Out-of-Core Methods

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Gary L. Rogers Jr.

August 2011

© 2011by Gary L. Rogers Jr.
All Rights Reserved.

Dedication

I would like to dedicate this work to

my amazing wife, Anne Marie

my loving parents, Gary and Vera

and my supportive family, Desiree, Rodney, and Audrey.

Acknowledgements

First and foremost, I would like to thank my advisor, Dr. Michael Langston, for his continued support, guidance, and friendship during my years with the Langston Lab.

I would also like to thank my committee members, Dr. Jian Huang and Dr. Michael Berry, for their support and instruction with this dissertation. I would like to thank Dr. Brad Vander Zanden for his guidance and willingness to serve on my committee. I would like to extend a special thanks to Dr. Brynn Voy and Dr. Arnold Saxton for their many years of leadership and support while working on projects together.

I would like to thank all of the students that helped me over the years including Dr. Andy Perkins, Dr. John Eblen, Dr. Yun Zhang, Dr. Faisal Abu-Zhazam, Dr. Rachel Lynch, Charles Phillips, Sudhir Naswa, Jeremy Jay, Bhavesh Borate, Zuopan Li, Jordan Lefebvre, Dinesh Weerapurage, Clinton Nolan, Ron Hagan, Kai Wang, and Vivek Philip.

I would like to thank my colleagues from Sweden, Dr. Mikael Benson and Fredrik Barrenäs. I would like to extend my most sincere gratitude to Suzanne Baktash for everything over the years.

Finally, I would like to thank my family for their continued support and guidance throughout my life.

Many men go fishing all of their lives without knowing it is not fish they are after.

- Henry David Thoreau

Abstract

Biological data derived from high-throughput microarrays can be transformed into finite, simple, undirected graphs and analyzed using tools first introduced by the Langston Lab at the University of Tennessee. Transforming raw data can be broken down into three main tasks: data normalization, generation of similarity metrics, and threshold selection. The choice of methods used in each of these steps effect the final outcome of the graph, with respect to size, density, and structure. A number of different algorithms are examined and analyzed to illustrate the magnitude of the effects.

Graph-based tools are then used to extract putative gene networks. These tools are loosely based on the concept of clique, which generates clusters optimized for density. Innovative additions to the paraclique algorithm, developed at the Langston Lab, are introduced to generate results that have highest average correlation or highest density. A new suite of algorithms is then presented that exploits the use of *a priori* gene interactions. Aptly named the anchored analysis toolkit, these algorithms use known interactions as anchor points for generating subgraphs, which are then analyzed for their graph structure. This results in clusters that might have otherwise been lost in noise.

A main product of this thesis is a novel collection of algorithms to generate exact solutions to the maximum clique problem for graphs that are too large to fit within core memory. No other algorithms are currently known that produce exact solutions to this problem for extremely large graphs. A combination of in-core and

out-of-core techniques is used in conjunction with a distributed-memory programming model. These algorithms take into consideration such pitfalls as external disk I/O and hardware failure and recovery.

Finally, a web-based tool is described that provides researchers access the aforementioned algorithms. The Graph Algorithms Pipeline for Pathway Analysis tool, GrAPPA, was previously developed by the Langston Lab and provides the software needed to take raw microarray data as input and preprocess, analyze, and post-process it in a single package. GrAPPA also provides access to high-performance computing resources, via the TeraGrid.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Microarrays are the foundation of many biological experiments designed to identify and extract putative networks under a given set of conditions. The current trend in microarray design and development is to generate chips that have an increase in the number of measuring capabilities while decreasing the overall price per chip. Analyses of these larger and more abundant datasets require both advanced algorithms and hardware design. Currently, microarray datasets can be analyzed using a wide variety of software ranging from complex commercially available tools to open source statistical packages. The algorithm complexity used by these tools can diverge as much as the tools themselves, starting with approximation algorithms and culminate with exact algorithms. In addition to the complexity of the algorithms used, other factors that influence the results of the experiment include the experimental design, sample preparation, and the physical microarray chip, just to name a few.

Another important consideration in the analysis of high-throughput microarrays is the computational requirements of the analysis. Most software currently in use is designed to run sequentially on a single machine. However, the size and complexity of the datasets being produced by state-of-the-art technologies will soon require the use of high-performance computing systems and parallel algorithms to complete the analysis of experiments in a timely and efficient manner.

1.1 Motivation

The focus of this dissertation is the analysis of biological datasets using a combination of out-of-core based tools and high-performance hardware and software. Graph-based algorithms have been used to analyze high-throughput microarray data [1, 2, 3] and extract meaningful putative biological networks [1, 4, 5]. These putative biological networks are produced by clique-centric algorithms [6, 7]. These computationally intensive algorithms introduce complexities to the analysis of large datasets, such as requiring the concurrent storage of the entire graph in core memory. Other important factors to graph-based high-throughput microarray analysis include the selection of normalization methods, threshold levels, and similarity metrics. Graph-based algorithms allow for parameters to be tuned to account for the inherent pitfalls in the data, such as noise.

Large upfront investments are required when generating biological data, with respect to both time and money. Therefore it is imperative for scientists to have access to the correct tools necessary to analyze data precisely and efficiently. The combination of both high-performance hardware and software not only make it possible to generate exact solutions efficiently, but make the analysis of larger and more complex datasets a reality. Exact solutions to problems, such as clique, allow for scientists to get the most accurate results possible from the analysis of their datasets.

1.2 Microarray Basics

Microarrays are used to measure many different biological properties including gene expression, single nucleotide polymorphisms (SNPs), and alternative splicing [8]. Many publications have presented work derived from microarrays in case/control studies[9], or in time-series analyses [10]. The results of the microarray experiments range from the identification of differentially expressed genes [11] to the discovery of

SNP association in complex diseases [12]. The remainder of this dissertation refers to microarrays that measure gene expression levels, unless otherwise noted.

1.2.1 Background

Microarrays have been an invaluable tool for biologists in the past decade and a half. In 1995, Mark Schena et al. reported the first microarray experiment results in *Science*. The experiment measured the expression level of 45 genes in the small flowering plant, *Arabidopsis thaliana*, in both a wild-type and a transgenic line overexpressing the single transcription factor HAT4 [13]. The importance of this new type of experiment can be seen by the nearly 7,000 citations it has accrued over the years and by the number of published microarray analyses. The surge of new microarray data has spawned online data repositories, such as Gene Expression Omnibus (GEO) [14], in order to allow scientists to publish and share their datasets. There are currently* 23,879 publicly available datasets on GEO, comprising of 9,021 different microarray platforms, 23 different microarray vendors, 16 different species, and 592,552 different samples across all datasets. The number of datasets has been growing at an average rate of nearly 47% over the past 6 years, as seen in Figure 1.1.

1.2.2 Data Generation

The basic design of microarray chips is essentially the same across all technologies and vendors. A typical microarray chip is a large collection of hybridization probes attached to a solid surface. The length and number of these hybridization probes differ between chips and vendors [15]. A single hybridization probe is typically a collection of oligonucleotides arranged in a very specific order so that it bonds only with the the complementary RNA (cRNA) of the segment of messenger RNA (mRNA) that is to be measured. Each hybridization probe measures a defined section of a gene, and a collection of probes, defined as a probe set, measures the

*As of July 13, 2011

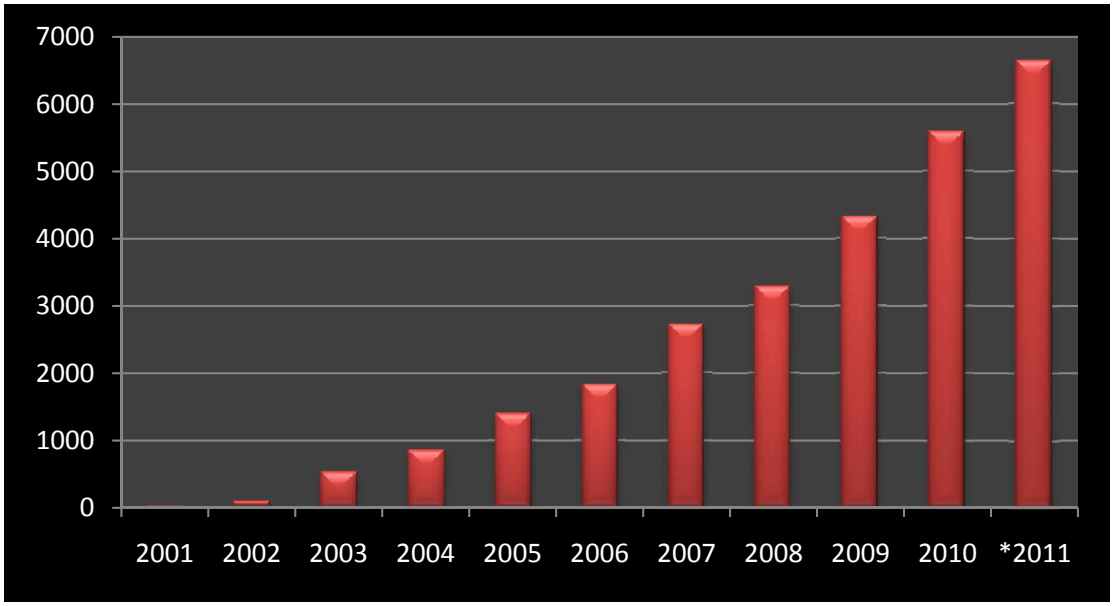


Figure 1.1: Number of datasets in GEO, listed by number of uploads per year. GEO started in 2001 with 13 datasets and has grown at a steady rate over the past decade. It is projected to have over 6,500 uploads in 2011 alone.

expression level of a single gene. Overall, the process of extracting mRNA from a specimen, reverse transcribing, labeling with a special fluorescent dye, hybridizing to the microarray chip, and measuring the intensity of the fluorescent signal is fairly well standardized across all platforms. However, this process of generating data does allow for many opportunities for variation to be introduced into the experiment and must be accounted for. Different normalization methods are discussed later in this dissertation to account for technical variation, or noise.

Microarray Vendors

Microarray vendors have incorporated many different technological designs into their respective chips. Although there are more than 23 different microarray vendors currently listed in GEO, only two of the largest vendors will be surveyed, Affymetrix [16] and Illumina [17]. Affymetrix uses a chip design that places the hybridization probes in a systematic manner across the entire chip and the location of the probes are known at the time of manufacture. For each target sequence, there exists a perfect

match (PM) and a mismatch (MM) probe. The Affymetrix probes are 25-mer in length, with the PM probe being the exact compliment of the mRNA, cRNA, being measured. The MM probe differs from the cRNA at the 13th position, as illustrated in Figure 1.2. Some normalization methods, such as MAS 5.0, use the PM/MM pairs in the computation of the gene expression level, while other methods, such as RMA [18, 19], disregard the MM probes completely, thus rendering half of the Affymetrix chip useless. Positive aspects of the Affymetrix chip include a large user base, numerous package support and consistently precise results [20]. Some of the drawbacks of the Affymetrix chip includes identification of SNPs at the 13th position of the MM probe [21], redundant probe sets, and the large amount of mRNA needed to measure gene expression levels [22].

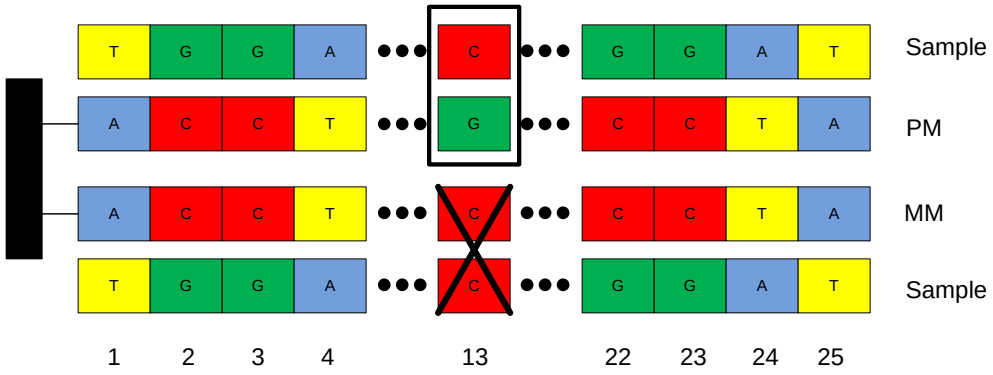


Figure 1.2: Affymetrix incorporates a Perfect Match (PM) probe along with a Mismatch (MM) probe design. Both probes are 25-mer in length and differ at the 13th position.

Illumina uses a different approach when attaching the hybridization probes to the chip. Instead of attaching the hybridization probes in a systematic manner, the probes are attached to silica beads and the beads are then randomly distributed among the wells on the substrate. The total length of the Illumina hybridization probes are 79-mer. The 29-mer on the 3' end is used by Illumina for identification of the probe, and the 50-mer on the 5' end is used for measuring mRNA expression level, see Figure 1.3. Illumina does not incorporate the PM/MM strategy for each target sequence, allowing for more hybridization probes on the chip. Advantages for the Illumina technology

include the ability to run more chips at a single time, generally less expensive and it requires a smaller amount of total mRNA to generate gene expression levels [22]. However, some of the drawbacks to this technology include a small, but growing, user base, limited software support outside of BeadArray, and less precise than Affymetrix [23].

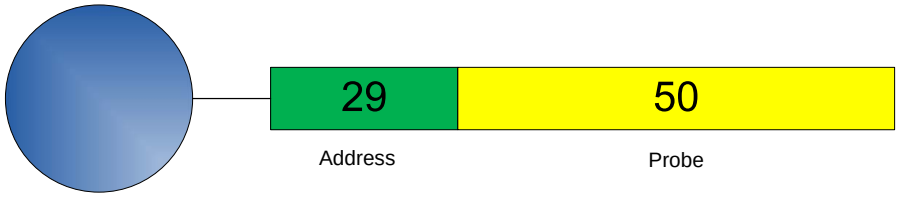


Figure 1.3: Illumina probes are 79-mer long and are divided into two sections: the 29-mer address section that is for identification of the probe and a 50-mer probe section.

Both technologies suffer somewhat from changes in annotation in the ever evolving field of bioinformatics. The probe sequence used in an Affymetrix chip to measure a gene is not guaranteed to be the same probe sequence used in the Illumina chip to measure the same gene. Even using different generations of the same chip design isn't guaranteed to use the same sequence to measure the same gene, which could lead to differences in expression levels when comparing two different experiments [24]. It is currently estimated that there are approximately 20,000 to 25,000 genes in the human genome [25], however, coverage of the whole genome is fairly good by both technologies, measuring around 50K probes each in the Human arrays. Given the technical aspects of both technologies are generally acceptable, the main forces behind chip selection for an experiment is still driven by how comfortable the end user is with the technology and the total expense in generating and analyzing the data.

1.2.3 Experimental Design

Microarray chip selection is only one of many factors in the experimental design. While the use of two-color microarrays can reduce the number of overall microarray

chips required for the experiment, the two-color microarrays also add a level of complexity to the analysis. The drawbacks of using two-color microarrays include an increase of data loss due to chip failure, the increase of the effect of an outlier sample, and a more complex experimental design (eg. saturated design vs single reference design as seen in Figure 1.4). The use of one-color microarrays require twice the number of microarray chips needed to perform the experiment, however, the chips are more robust to the effects of outliers, as only the outlier chip will need to be removed from the analysis. This does not effect the results of the other chips. One-color microarrays do not suffer from dye bias and the data can be easily compared to other arrays from other experiments [9]. The decrease in prices for one-color microarrays make them an attractive and viable choice for most experiments and the analyses in the remaining parts of the dissertation focus solely on one-color chip experiments. Other experimental design factors include the total number of samples, including the number of technical and biological replicates, and whether or not samples must be pooled together to generate enough biological matter to complete the experiment.

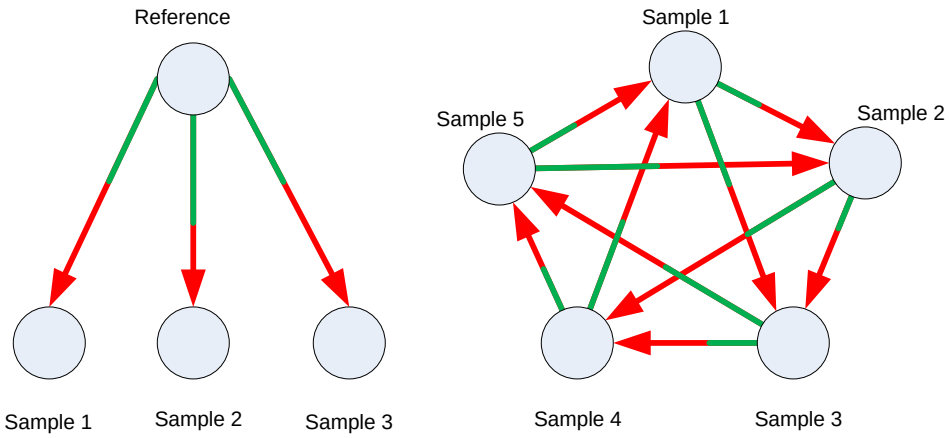


Figure 1.4: Two-color microarray experimental designs range in complexity. The experimental design on the left uses a single reference while the experimental design on the right uses a saturated model.

1.2.4 Preprocessing and Statistical Analysis

Once the experimental design is finalized and the biological data has been processed, the analysis of the data is ready to proceed. The first step in the analysis is preprocessing the RAW data. There are many software packages available, for either Illumina or Affymetrix data, such as BeadArray, R [26] with Bioconductor [27], and Flexarray [28]. R is an open source statistical software package with great user support for bioinformatics and is used for remainder of the analyses in this dissertation, where applicable. Preprocessing the data includes reading in all of the RAW data files, normalizing the data, and then assigning expression levels to the probes. The expression level of a probe is a function of the luminosity of the probe, as illustrated in Figure 1.5.

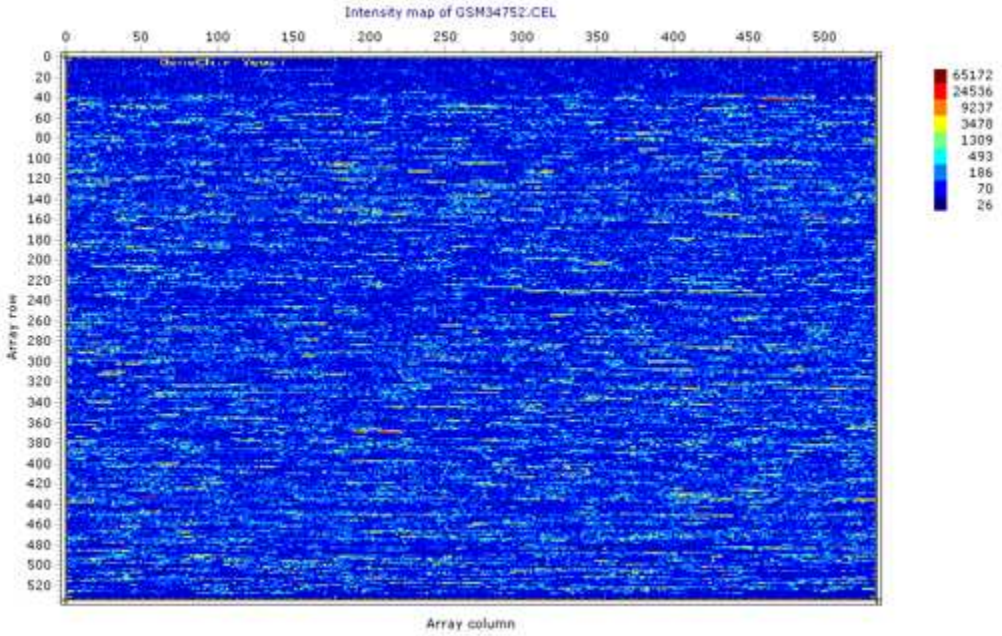


Figure 1.5: Affymetrix CEL file.

Algorithms

The normalization step in the analysis is necessary to correct for effects of variation in the microarray technology rather than true biological variation between the samples

[29]. Figure 1.6 illustrates the same dataset before and after normalization was applied. Many different preprocessing algorithms have been proposed for Affymetrix chips, such as MAS 5.0 [27], RMA [30], and dChip [31], and are discussed in detail in Chapter 3. Popular preprocessing algorithms for data derived from Illumina chips include average, rank invariate, and cubic spline normalizations. It is believed that different preprocessing algorithms leads to different results [32], and it isn't the case that one method always works best on all datasets.

Statistical tests

Basic statistical tests can be applied to identify genes that have a significant impact to the experiment. The standard statistical method for identifying the set of genes that are differentially expressed between two datasets of equal size and variance is the t-test. If the experiment has two or more conditions, the analysis of variance (ANOVA) model is the most appropriate to use [33]. T-tests and ANOVA models have been used extensively in the identification of differentially expressed genes with respect to microarray experiments [34]. The differentially expressed genes are assigned a p-value, which is the probability of obtaining a test statistic at least as extreme as the one that was observed, if H_o is true. In other words, how significant are the changes in the mean expression levels of a gene in the two groups. Given that thousands of probes are being tested, it is important to correct for multiple comparisons using a method such as Bonferroni correction or False Discovery Rate (FDR) [35]. Bonferroni correction is the simpler method and is defined as

$$p_{adjusted} = \frac{p}{n} \tag{1.1}$$

where p is the original p-value of a gene and n is the total number of tests. However, Bonferroni correction is often too conservative for microarray analysis. FDR limits the expected proportion of Type I errors resulting from the multiple tests. Much like p-values, there exists q-values which measures the minimum false discovery rate

when a test is deemed significant [36]. The results of these statistical tests are used in conjunction with the graph-based tools in order to winnow lists to include only the most significant genes [1].

1.3 Clique-centric analysis

A clique-centric analysis has been shown to be effective when analyzing microarray data [1, 2, 3]. Current microarray chips produce data that is on the order of tens of thousands of probes, when measuring for gene expression data. Extracting viable putative networks from datasets of this size is a perfect match for graph algorithms. The use of these tools to produce these networks is illustrated in Figure 1.7.

The primary step in our analysis is converting the data generated by microarrays into a graph. A graph $G = \{V, E\}$ is defined as a set V of *vertices* and a set E of *edges*. For the purpose of this dissertation, only simple, finite and undirected graphs are considered, see Figure 1.8. The conversion of the microarray data into a graph transforms the set of probes into vertices and assigns a weight to all pairwise edges. The weight assigned to an edge is generally denoted by the interaction between the two vertices, for example the correlation between two vertices. There are multiple similarity metrics from which to choose, as listed in Table 1.1, and the differences between these metrics are examined more closely in Chapter 3.

Clique-centric algorithms

Once the microarray data has been transformed into a complete weighted graph, it is true that

$$\forall u, v \in V, \exists \{u, v\} \in E, \text{ such that } \omega(u, v) \text{ is defined}$$

where ω is the weight function. It is necessary to apply a threshold to the edges in our graph and retain only the putatively significant edges and vertices. This filtering step also transforms the graph from a weighted graph into an unweighted graph. Many

Similarity Measure	Range	Type	Formula
Pearson	[-1,1]	Correlation	$\frac{\sum(X_i-\overline{X})(Y_i-\overline{Y})}{(n-1)S_xS_y}$
Spearman	[-1,1]	Correlation	$1-\frac{6\sum d_i^2}{n(n^2-1)}$
Kendall Tau	[-1,1]	Correlation	$\frac{n_c-n_d}{\frac{1}{2}n(n-1)}$
Mutual Information	> 0	Correlation	$\sum_y\sum_x p(x,y)\log\left[\frac{p(x,y)}{p1(x)p2(y)}\right]$
Manhattan Distance	> 0	Distance	$\sum\lvert X_i-Y_i\rvert$
Euclidean Distance	> 0	Distance	$\sqrt{\sum(X_i-Y_i)^2}$
Jaccard Index	[0,1]	Association	$\frac{\lvert X\cap Y\rvert}{\lvert X\cup Y\rvert}$

Table 1.1: Collection of similarity measures.

threshold selection methods have been proposed [37, 38], and several methods are reviewed in depth in Chapter 3. The threshold selection and filtering steps typically reduce the size of the graph into a more manageable graph, however, extracting the densest subgraphs, or networks, from the graph is still very computationally intensive. By definition, the densest possible subgraph is a clique. A clique is defined to be the set of fully connected vertices in a graph, with a density of 1, and it is known to be an NP-complete problem [39]. The extraction of cliques from a graph is sometimes feasible due to the complimentary dual of clique, vertex cover [40]. Vertex cover is the set of vertices that cover all edges, and it is known to be fixed-parameter tractable (FPT) [41]. A problem is said to be FPT if given an input of size n and a parameter k, there exists an algorithm to solve the problem in $O(f(k)n^{(c)})$ time. FPT algorithms also reduce the problem to a kernel, which allows for an even smaller instance size to be solved. The relationship between clique, vertex cover, and FPT are discussed in more detail in Chapter 2.

The cliques of a graph are the core structures of the biological networks derived from the microarray data. However, the requirements for clique that every edge be present is sometimes too restrictive. There are many points along the way in the analysis that could allow for a putatively significant edge to be excluded from the

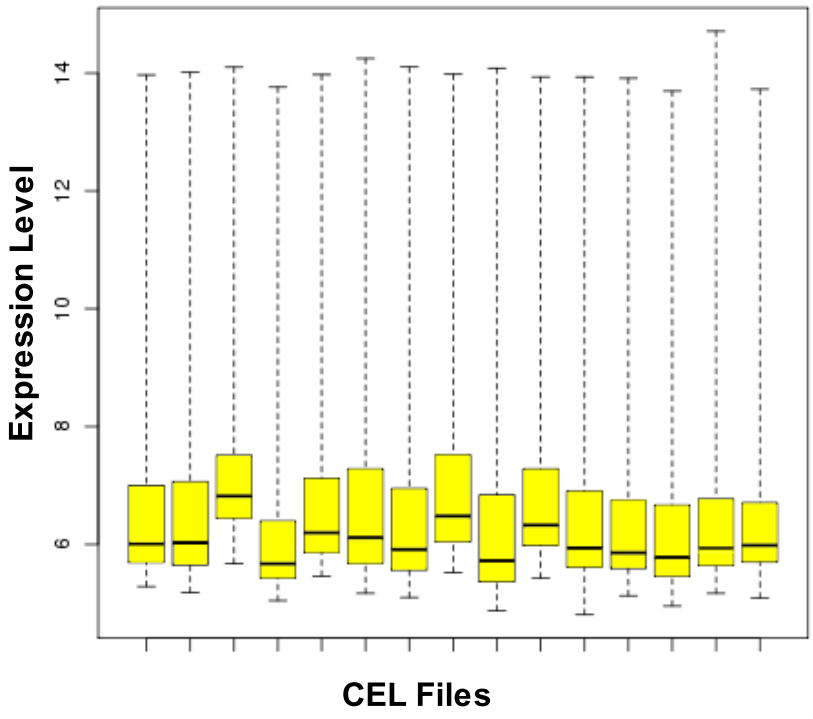
analysis, such as threshold selection, or noise inherent in the data has influenced the analysis in such a way that a few key edges have been eliminated from our analysis. The use of soft-thresholding is one method to try to resolve these missing edges that, biologically speaking, should be included in our analysis. The paraclique algorithm, developed by M. A. Langston, is one such example of a soft-thresholding algorithm. These clique-centric algorithms produce dense putative biological networks. These networks can then be studied independently, or in parallel to determine difference between the networks.

Graph-based differential algorithms, similar to differential expression, include differential correlation and differential topology [1]. These algorithms can be used to identify genes that belong to a response network of a stimulus or identify the genes that interact with a completely different network under a given condition. Genes that have been identified in previous steps or other analyses can be used to refine our analysis by using methods such as anchored clique, anchored paraclique and anchored biclique. A review of these algorithms are presented in Chapter 4.

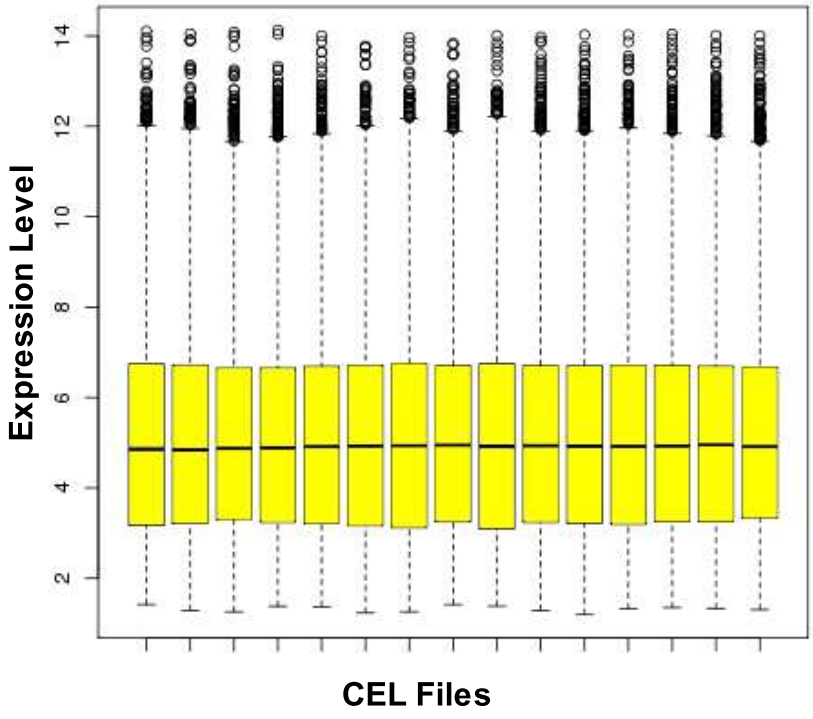
All of the datasets and algorithms up to this point have assumed that the data fits into core memory on a single machine. This is not always the case, for example, with datasets measuring SNPs. It is estimated that there are 10 million SNPs in the Human genome [42] and current Illumina SNP chips measure 2.5 million SNPs, with the real estate on the chip readily available to measure up to 5 million SNPs. Due to the data structure requirements of current tools, datasets of this size are simply too large to fit into core memory on most machines. Therefore it is imperative that parallel out-of-core algorithms be implemented. The biggest concern with out-of-core algorithms is the amount of time spent doing I/O and making external passes over files stored on disk. These and other items are examined in closer detail in Chapter 5.

1.4 Postprocessing

The postprocessing step in the analysis can include visualization, validation of results, and to ascertain the function of unknown genes. Visualization of the data allows the results to be analyzed for network structure and for identification of key genes in the network. Graphviz [43], a well-known open-source visualization package, is available from AT&T labs. This particular software package allows for the results to be visualized in a number of different layouts, depending on the type of graph provided. Figure 1.9 illustrates one of the layouts plotting non-overlapping maximal cliques in a graph. The results generated can also be validated using a wide array of tools. These tools range from open source software, such as Cytoscape [44], DAVID [45] and Gene Ontology [46], to subscription based software such as Ingenuity Pathway Analysis [47]. These tools rely on current biological knowledge, usually derived from current publications, either in an automated manner using text mining tools, or in a manually curated manner, such as Ingenuity. These tools can be used to determine if the members of the generated networks are significantly enriched in certain categories or pathways. Genes in these generated networks could have very little functional information published. One could ascertain this functional information by combining the network produced using clique-centric tools and the enrichment information of the other members in the network. In this manner, new functional information can be proposed and further studied in a wet lab environment. Figure 1.10 is an example of a KEGG pathway visualization produced by DAVID.



(a)



(b)

Figure 1.6: The effects of normalization are illustrated using the pre-normalization data (a) and the post-normalization data (b).

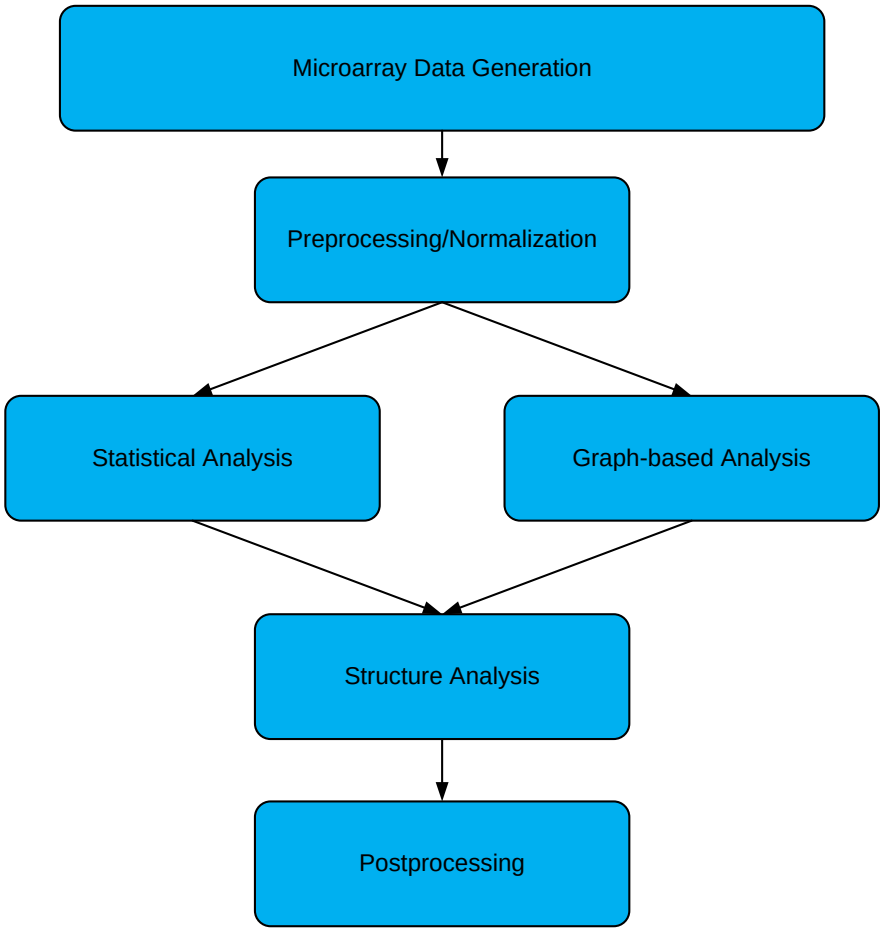


Figure 1.7: High-throughput microarray analysis from data generation to postprocessing. A combination of clique-centric tools and statistical tests extract only the most putatively significant networks.

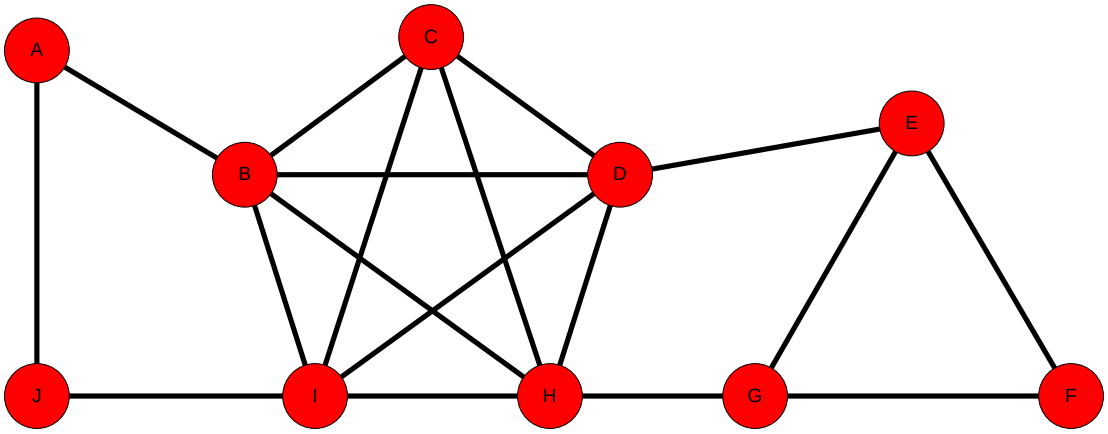


Figure 1.8: An example of a simple, finite, undirected graph. There are 10 vertices and 18 edges. This graph will be used as an example graph throughout the rest of the dissertation.

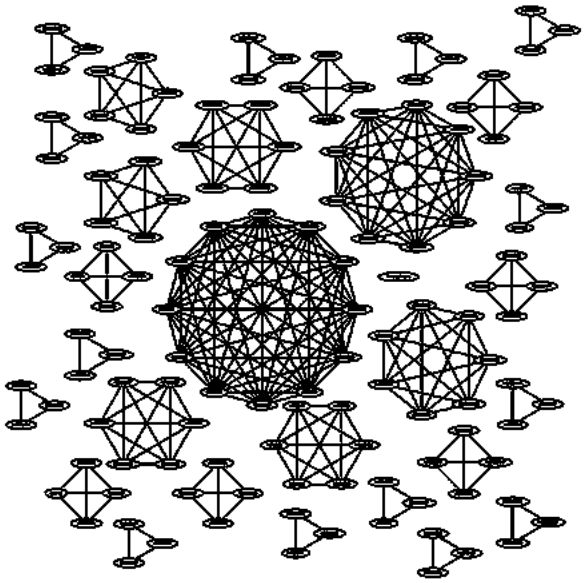


Figure 1.9: An example of non-overlapping maximal cliques in a graph. It is typical to see few very large maximal cliques and numerous smaller maximal cliques in biological data.

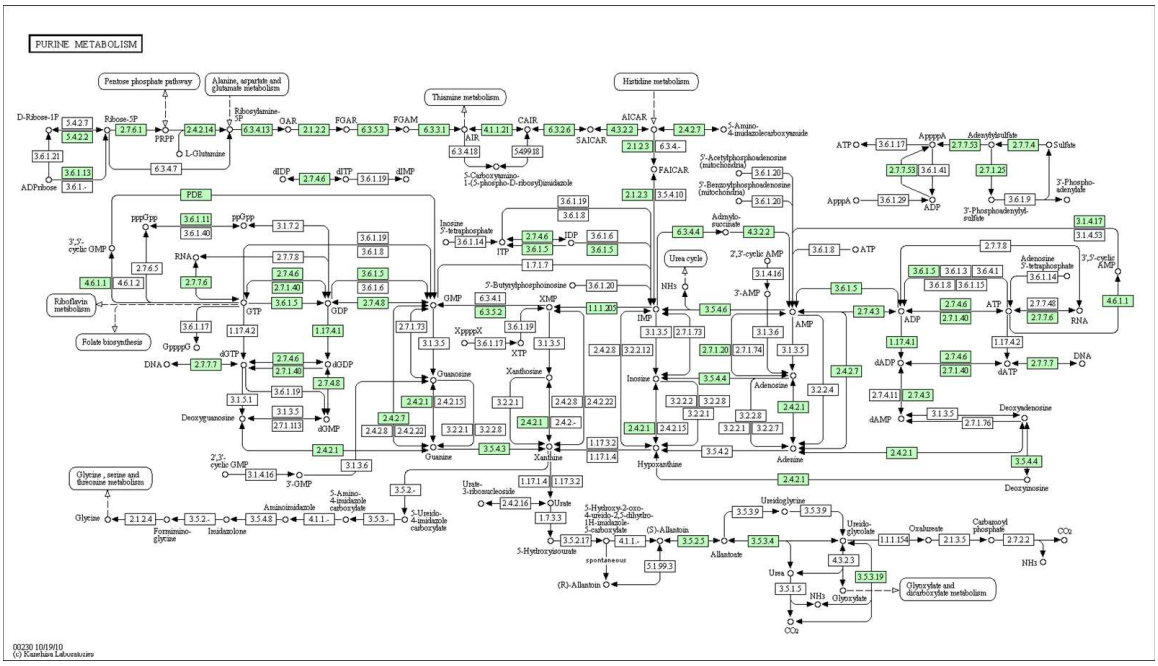


Figure 1.10: A KEGG pathway generated by the DAVID Bioinformatics tool.

Chapter 2

Graph-based Tools Overview

2.1 Introduction

Everything can be represented as a graph. This statement is the motivation behind many centuries of exploring graph theory. In the context of this research, the interaction of biological elements is represented as a graph and analyzed using properties of inherent structures, namely clique-centric structures. Generally speaking, this research focuses on the interaction between transcriptomic data, however, it is important to note that other types of -omic (genomic, proteomic) data can easily be substituted in any of the analyses once the data has been transformed into a graph. The extraction of dense subgraphs using clique-centric based tools proves to be a successful method of modeling real biological data [1, 3, 5, 6]. This section will introduced the basic notation and definitions needed to successfully parse the remainder of this dissertation, followed by basic graph problems such as Vertex Cover and Clique.

2.2 Definitions

All graphs, unless specifically stated otherwise, are considered to be simple, finite, and undirected graphs. A graph $G = \{V, E\}$ is defined as a set V of *vertices* and

a set E of *edges*. An edge is a set of vertices, $\{u, v\}$. A *subgraph* of G is defined as $G' = \{V', E'\}$ where $V' \subset V$ and $E' \subset E$. A graph is defined to be *complete*, K_n, if $\forall\ u, v \in V$, $\exists\ \{u, v\} \in E$. In other words, all possible edges are present in the graph. The *density*, $D(G)$, of a graph is $\frac{2|E|}{|V|(|V|-1)}$, where $0 \leq D \leq 1$. Vertices u, v are said to be *adjacent* if $\{u, v\} \in E$. The set of vertices adjacent to a vertex, v are the *neighbors* of v and constitute the *neighborhood* of v, $N(v)$. Any vertex not adjacent to v belongs to $\overline{N}(v)$. The *union* of these two sets, $N(v) \bigcup \overline{N}(v)$ is equal to $V - v$. The *degree* of vertex v, $d(v)$ is $|N(v)|$, where $0 \leq d \leq (|V| - 1)$. Vertex v with $d(v) = 0$ is said to be *isolated*. Vertex, $v' \in G'$ with $d(v') \approx |V'|$ is said to be a *star node*. A *path*, $p(u, v)$ is the set $\{\{u, w_1\}, \{w_1, w_2\}, ..., \{w_k, v\}\}$, where $w_{1..k}, u, v \in V$ and $\{\{u, w_1\}, \{w_1, w_2\}, ..., \{w_k, v\}\} \in E$. This simply states that v is reachable from u. A *connected component*, CC, is the set of vertices such that $\forall\ u, v \in CC$, $\exists\ p(u, v) \in E$. A graph with a single connected component is said to be *connected*; otherwise it is considered *disconnected*. The complement of G is $\overline{G} = \{V, \overline{E}\}$ where $\overline{E} = [V]^2 - E$.

2.3 Vertex cover

In 1972, Karp introduced a list of 21 $\mathcal{NP}$-complete problems [48]. Many of these problems are still extensively studied today, including vertex cover.

Definition 1. *Given a graph $G = \{V, E\}$, a* vertex cover *is a set $VC \subseteq V$ such that $\forall$ u,v $\in E$, $((u \in VC) \vee (v \in VC))$.*

Like most computational problems in graph theory, there are two well-studied flavors of the vertex cover problem: the decision problem and the optimization problem. The decision problem is defined in the usual way:

Input: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Question: Is there a $V' \subseteq V$ for which $|V'| \leq k$ such that every edge in E at least

one endpoint is V'.

Figure 2.1 shows an example vertex cover. It is well-known that the decision version of vertex cover is $\mathcal{NP}$-complete and that the optimization version is $\mathcal{NP}$-hard [39]. This dissertation focuses on the optimization version of the vertex cover problem as defined below:

Input: A graph $G = (V, E)$.
Question: What is the smallest k such that G has a $VC \subseteq V$ of size k.

Although the optimization version of vertex cover is $\mathcal{NP}$-hard, and the expected algorithms used to solve such problems must have exponential run times in terms of the input size, using vertex cover in everyday analyses is feasible due to it also being FPT [49]. A problem is said to be FPT if there exists an algorithm to solve it in $O(f(k)n^c)$, where c is constant. Since the run time of the algorithm is no longer dependent on the size of the input n, but rather the parameter k, solutions to FPT algorithms are more feasible. Also, certain advantages are gained with an FPT algorithm, namely the reduction of the problem to a kernel, as defined below by [50].

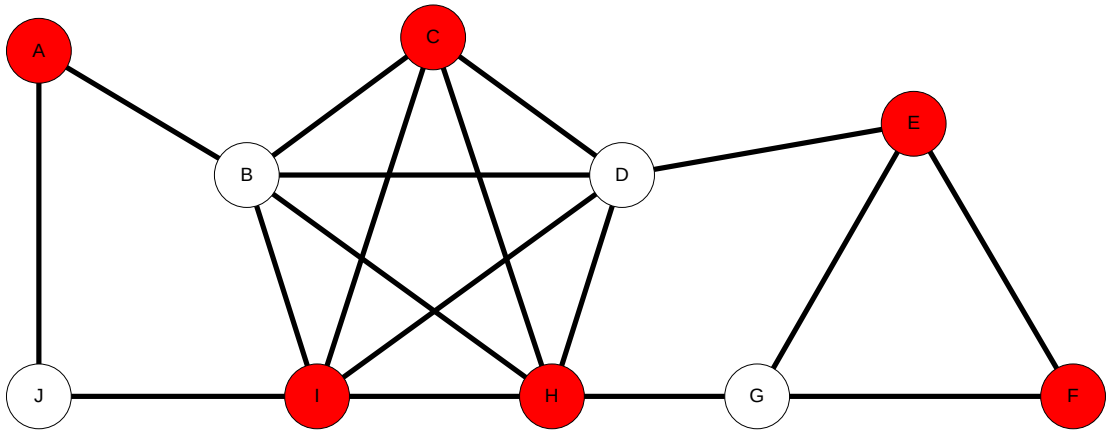


Figure 2.1: A vertex cover of our basic graph. The cover is {A,C,E,F,H,I}. While this cover is optimal, it is not unique.

Definition 2. *Given a parameterized problem P with the input pair (I,k), where I is the problem instance and k is the parameter. Reduction to a problem kernel replaces the original instance (I,k) with (I', k') such that*

$$k' \leq k, \text{ and } |I'| \leq g(k)$$

for some function g depending solely on k, and

$$(I, k) \in P \text{ iff } (I', k') \in P.$$

Also, the reduction from (I,k) to (I', k') must be computable in polynomial tim $T_K(|I|, k)$ The function g(k) is called the size of the problem kernel.

This simply states that if a problem is FPT, then there must exist a problem kernel, which is a reduced instance of the original problem. Conversely, if there exists a problem kernel, then the problem is FPT. Therefore, there exists a problem kernel for vertex cover. Many kernelization methods have been proposed for vertex cover [50, 51], however, the kernelization method used in this dissertation is based on crown reduction. The most straightforward crown reduction is the 1-degree rule, where the neighbors of vertices that have degree 1 are placed into the crown. It is straightforward to see that if $d(v) = 1$, then placing $N(v)$ into the crown from the graph yields at least the same result as placing v into the crown. The worst case only occurs when $d(N(v)) = 1$. In the case where $d(N(v)) > 1$, including $N(v)$ in the crown also covers all edges adjacent to $N(N(v))$. The high degree rule simply states that if $d(v) > k + 1$, then v must be included in the crown. Suppose v was not included in the crown, then all $N(v)$ must be in the cover; however, it is known that $|Nv| > k$, and thus a cover of size k cannot exists. Therefore, v must be in the crown for a cover of size k. The most trivial reduction in the graph is 0-degree rule. Simply stated, any isolated vertex ($d(v) = 0$) is removed from the graph. The combination of these three rules reduces the graph to at most $k^2 + k$ vertices and retains at most k^2 edges and the best known bound on vertex cover is $O(1.2852k + kn)$ [50].

2.4 Clique

Clique also belongs to the list of 21 $\mathcal{NP}$-complete problems introduced by Karp [48].

Definition 3. *Given a graph* $G = \{V, E\}$*, a* clique *is a set* $C \subseteq V$ *such that* $\forall$ *u,v* $\in C$, $\{u, v\} \in E$.

Much like vertex cover, there is a decision version of the clique problem, defined below:

Input: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Question: Is there a $V' \subseteq V$ for which $|V'| \geq k$ and such that $\forall$ u,v $\in V$, $\exists\{u, v\} \in E$.

The optimization version of the clique problem seeks to determine the largest clique in the graph.

Input: A graph $G = (V, E)$.
Question: What is the largest k such that G has a $C \subseteq V$ of size k.

The astute reader will have recognized the close relationship between both vertex cover and clique. This is due to the fact that clique is the complementary dual to vertex cover. The relationship between clique and vertex cover is extremely important because clique is both $\mathcal{NP}$-complete and believed to be fixed-parameter *intractable* as clique is, in fact, $W[1]$-hard in the W-hierarchy [50]. In order to use the fact that vertex cover is FPT and that it is the complementary dual to clique, the initial problem instance must undergo a transformation, and this transformation relies on independent set.

Definition 4. *Given a graph* $G = \{V, E\}$*, an* independent set *is a set* $IS \subseteq V$ *such that* $\forall$ *u,v* $\in IS$, $\{u, v\} \notin E$.

As a brief overview of this transformation, the problem instance from vertex cover to clique is discussed. Vertex cover takes as input the pair (G,k) and a solution, VC, is generated. This problem instance is then transformed into an independent set by taking the complement of the vertex cover solution, $I = \overline{VC}$. Next, by taking the graph complement, $\overline{G} = \{V, [V]^2 - E\}$, the problem instance is transformed into a clique instance, $\{\overline{G}, |V| - k\}$. In other words, a vertex cover of size $|VC|$ in G is equivalent to a clique of size $|V| - |VC|$ in $\overline{G}$ and $VC = V/C$ and $C = V/VC$. It is also important to note that the solutions to the optimization versions are also transformable, in the sense that a minimum vertex cover can be transformed into a maximum clique, and vice versa [49].

2.4.1 Maximal Clique

Given that a clique C in G is a set of fully connected vertices, extremal clique properties such as maximal clique and maximum clique can be defined.

Definition 5. *Given a graph $G = \{V, E\}$, a maximal clique is a set $C \subseteq V$ such that $\forall$ u,v $\in C$, $\{u, v\} \in E$ and $C \not\subseteq C'$, where C' is another clique in G.*

Maximal cliques play an important role in analyzing biological data due to the fact that these are not strictly the largest cliques in the graph, but they are comprised of the most elements that have a high interaction with each other. Maximal cliques can overlap each other, so membership to one clique does not prohibit membership to another clique. In 1965, Moon and Moser [52] showed that a graph with n vertices can have, at most, $3^{\frac{n}{3}}$ maximal cliques. Algorithms that generate all maximal cliques are divided into two groups: iterative enumeration and backtracking. Iterative enumeration algorithms, such as the one proposed by Kose *et al* [53] must first build all maximal cliques of size $k - 1$ before generating any cliques of size k. Memory requirements are typically prohibitive in this type of algorithm since all maximal cliques must be stored in memory or on disk. Backtracking algorithms are derived from the work of Bron and Kerbosch [54]. These algorithms use a "depth first"

approach to building maximal cliques. A maximal clique is grown by supplementing the current clique with vertices from a candidate list. Once the candidate list is empty, a maximal clique is produced. The vertex added last is removed and the non-maximal clique selects a vertex from the remaining candidate vertices at this recursion level. The algorithm recurses all possible levels until the entire search space has been examined.

2.4.2 Maximum Clique

All maximum cliques are maximal cliques with the largest number of vertices. Figure 2.2 illustrates the difference between a maximal clique and a maximum clique. The clique number of a graph, $\omega(G)$, is the size of a maximum clique of G. There can be numerous maximum cliques in a graph. All maximum cliques can overlap, but they all must have the same number of vertices. Maximum cliques are important to biological data in the sense that they are the largest networks responding to a given condition or stimulus. By default, any algorithm that enumerates all maximal cliques also enumerates the list of all maximum cliques. It is straightforward to filter the enumeration of maximal cliques keep only the maximal cliques of maximum size. Algorithms that focus solely on enumerating maximum cliques have been proposed that exploit the clique number of a graph to converge to a solution [55]. Maximum cliques are also used as the input for other clique-centric algorithms discussed in this dissertation, such as the paraclique algorithm.

Definition 6. *Given a graph $G = \{V, E\}$, a maximum clique is a set $C \subseteq V$ such that $\forall$ u,$v \in C$, $\{u, v\} \in E$ and $|C|$ is maximum.*

2.4.3 Biclique

A graph that has two distinct partitions that allows interpartition edges, but disallows intrapartition edges, is a bipartite graph. Bipartite graphs are subject to the same graph property problems, such as vertex cover, clique, etc, but each problem instance

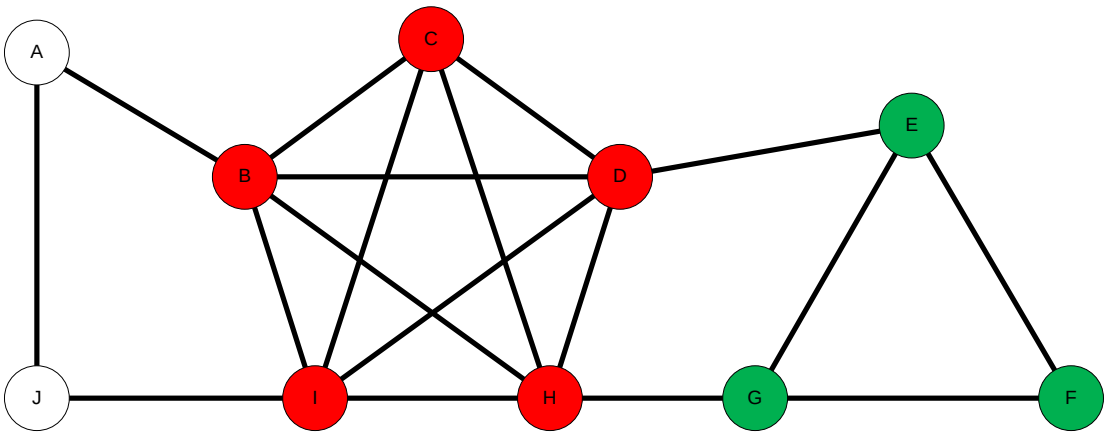


Figure 2.2: There are two maximal cliques in the graph. The green maximal clique is size 3 and the red maximal clique is size 5 and is a maximum clique in the graph.

must be tweaked to handle the format of the bipartite graph. For example, a bipartite graph cannot contain a clique of size 3 or greater, however, a biclique is defined as being a complete subgraph of a bipartite graph, as defined below.

Definition 7. *A bipartite graph $G = \{V, U, E\}$ has two disjoint sets, V and U, and a set of edges $\{u, v\} \in E$, where $u \in U$ and $v \in V$.*

In the case of a simple graph, all maximum cliques have the same number of vertices and the same number of edges. However, in a simple bipartite graph, bicliques can be either edge maximum, or vertex maximum, bicliques. Edge maximum bicliques are bicliques with the largest number of incident edges within the biclique, and vertex maximum bicliques have the largest number of vertices. Figure 2.3 illustrates the difference between the two.

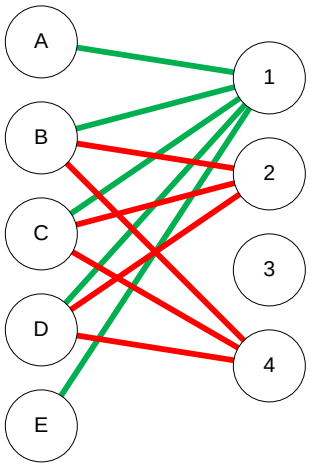


Figure 2.3: Two bipartite cliques are represented. The green bipartite clique is vertex maximal with 5 edges and 6 vertices, while the red bipartite clique is edge maximal with 6 edges and 5 vertices.

Chapter 3

Preprocessing and Graph Creation

3.1 Introduction

Selecting a preprocessing method, similarity metric and threshold selection algorithm can have a large influence on the properties of a graph. This chapter reviews some of the more popular algorithms and metrics to determine the effects each has on the properties of a graph. In the end, given any combination of each one of these algorithms and metrics will produce a simple, undirected, finite graph. The effects of each method are measured using *Saccharomyces cerevisiae* (baker's yeast) data generated on the Affymetrix microarray platform and properties of the graph, such as density and connectivity, are scrutinized.

3.2 Data and Software

The dataset was generated on the Affymetrix Yeast Genome S98 array. The dataset is publicly available for download from the GEO website under the series GSE1938 [56]. The open source statistical package, R version 2.11.1 [26], was used for all of the preprocessing and normalization steps, calculating the Mutual Information, and multiple testing correction. The qvalues were computed using Qvalue [36]. The Bioconductor package [27] and libraries were used when necessary to complete the

analyses using R. Datagen version 1.4a is custom software, written by Jon Scharff, was used to compute the Pearson, Spearman, and Kendall correlation metrics.

3.3 Preprocessing

Preprocessing data is usually completed using pre-built libraries that are publicly available through software packages such as R and Bioconductor. The most popular algorithms are listed in Table 3.1. These algorithms are compared using the values and distribution of the resulting expression levels.

Normalization Method	PM/MM	Background Subtraction
Robust Multi-Array Average (RMA)	PM	Lowest signals subtracted out
GC-Robust Multi-Array Average (GC-RMA)	PM	Incorporates non-specific bindings
GC-Robust Multi-Array Average (GC-RMA)	Both	Incorporates non-specific bindings
dChip	PM	Model based
dChip (w/MM)	Both	Mismatch subtraction
Microarray Analysis Suite 5.0 (MAS 5.0)	Both	Mismatch subtraction/Imputation

Table 3.1: A sample of the most popular normalization methods for the Affymetrix microarray. Some methods use the MM probes in the background correction step, while others ignore them completely.

Preprocessing algorithms have three basic functions: *background correction*, *normalization*, and *expression level quantification* [57]. Background correction is used in the removal of unwanted background signal. The methodology employed by the various algorithms range from the use of the MM probe signals, to the estimation of the background signal derived from the lowest level signals of the PM probes [18]. Affymetrix chips come with both PM and MM probes, and any preprocessing

algorithm that does not use MM probes are, by default, using only half of the data generated on the chip. However, in [30], it is shown that the MM signal increases as the PM signal increases. Therefore, using the MM signal value as the background correction metric, it is actually effecting the signal of the PM probe. The second function, normalization, is required to correct for effects of variation introduced by microarray hardware. A handful of normalization methods are employed by the algorithms, such as linear scaling and quantile normalization. Finally, quantifying expression levels include the combination of probe level signals into the expression level of a probe set, and these signals are typically log transformed to help facilitate analysis [58]. While most preprocessing algorithms have the same basic functionality, the details of each algorithm are reviewed:

- **RMA** - Assumes a signal model with both additive and multiplicative error components. This method disregards all MM probes and uses only the PM probes in determining both background signal and expression level values. The data is quantile normalized between arrays and log transformed.

- **MAS 5.0** - This method was distributed by Affymetrix for use with its hardware, and does exploit the MM probes when computing background signal. Linear scaling is used to ensure common distribution of the signals over the arrays; however, the data is generally not log transformed by the algorithm.

- **GC-RMA** - A derivative of RMA that incorporates a bias correction algorithm that estimates non-specific bindings [59]. Non-specific bindings are estimated using probe nucleotide sequences. Like RMA, the data is log transformed and quantile normalized.

- **GC-RMA (MM)** - Similar to the GC-RMA algorithm except that MM probe values are also used in calculating the background signal.

- **dChip** - One of the first model-based algorithms designed for microarray analysis [28]. This algorithm uses the invariant set, the set of probes whose

signal is uniform across all arrays, normalization method. Background signal is computed by splitting array into zones and computing background signal per zone.

- **dChip (MM)** - Same methodology as dChip, except incorporates the signal produced by the MM probes in the background correction.

3.3.1 Results

The Yeast dataset, comprised of 15 microarrays, was processed using all 6 algorithms. Figure 3.1 uses the expression levels produced on a single microarray to generate the expression level histogram. The histogram shows the distribution of the expression values. The top 30 expressed probes, resulting from the RMA normalization method, are highlighted in black. Note that these 30 probes are highly expressed in all of the preprocessing algorithms, as expected, but the list of the top 30 expressed probes are not identical across all preprocessing algorithms. This is demonstrated by the fact that the highlighted regions have a larger spread in the histogram of some algorithms when compared to others. The MAS 5.0 and the dChip (MM) are very similar in their expression patterns, while the other methods have fairly similar outputs. The use of the MM probes in the GC-RMA (MM) method does not seem to significantly influence the histogram, in this case.

Figure 3.2 shows the heatmap of the expression level of the same top 30 probes generated by the RMA method. Similar patterns are observed in the RMA and GC-RMA methods, which is to be expected since the GC-RMA is a derivative of the RMA method.

3.4 Similarity metrics

The subsequent step after preprocessing is the computation of all pairwise similarity values. With respect to analyzing microarrays, this is computing similarities between

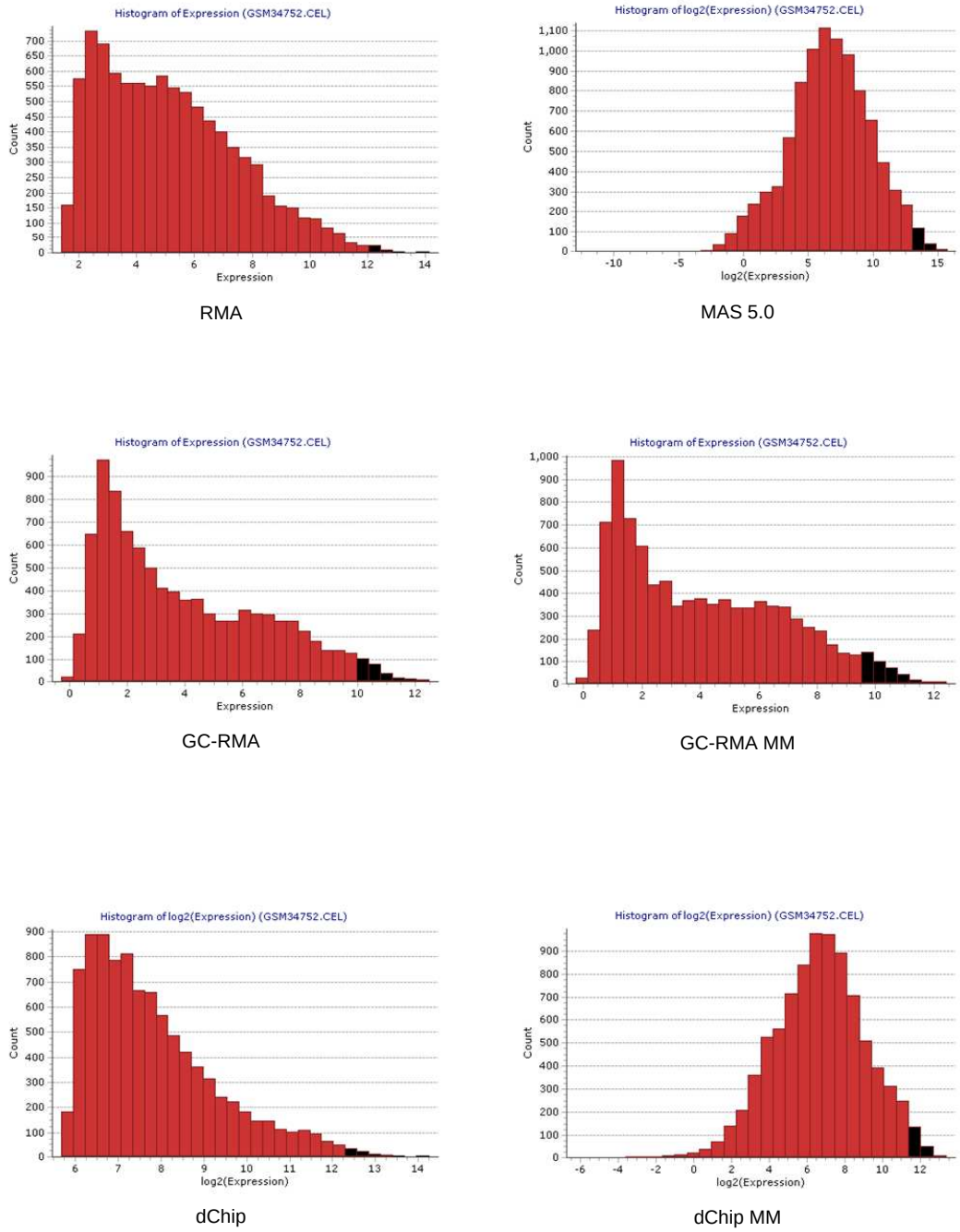


Figure 3.1: The expression levels of the Yeast data derived using the six different preprocessing methods. The top 30 expressed probes, identified using the RMA preprocessing algorithm, are highlighted in black in all six histograms.

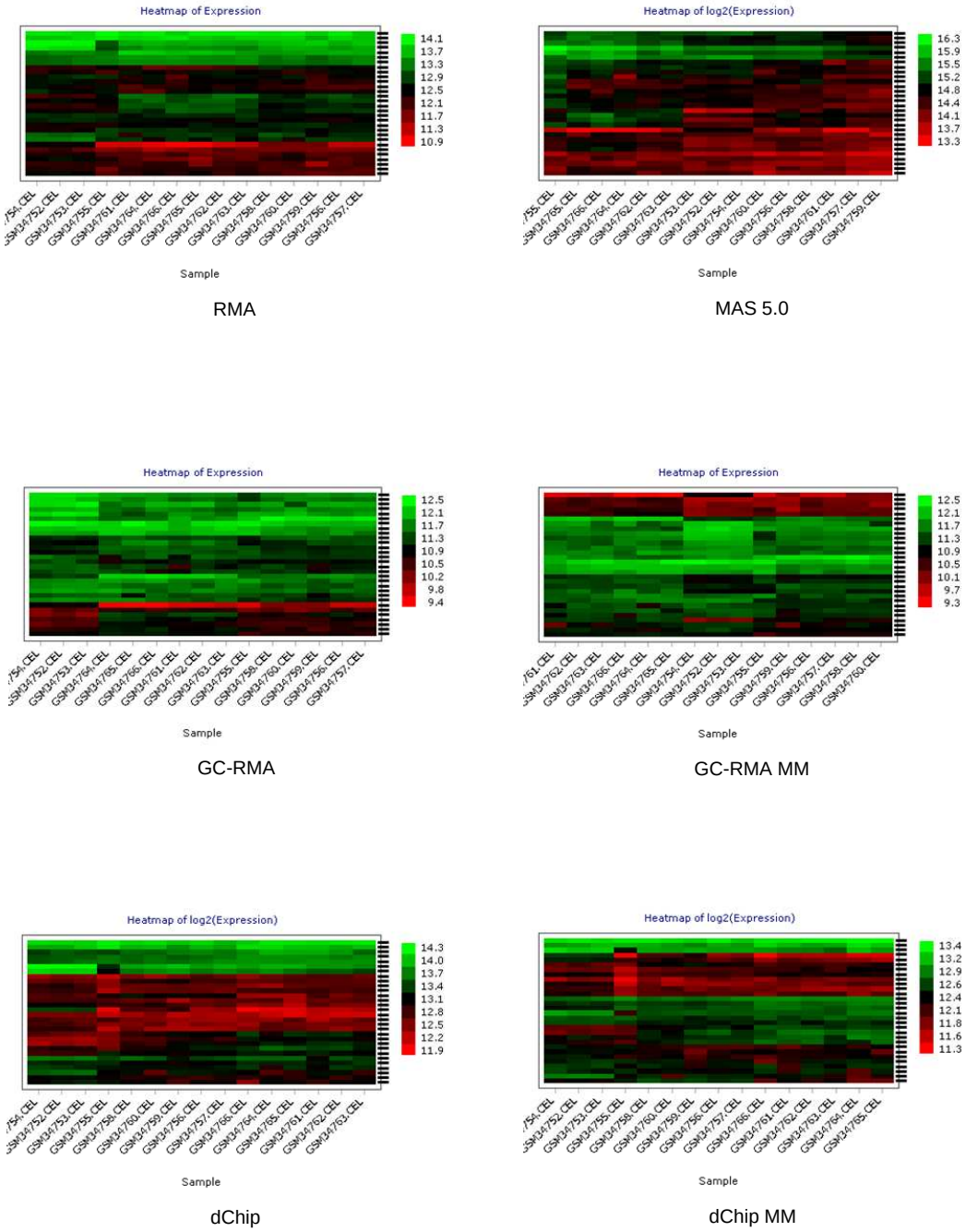


Figure 3.2: The heatmap of the expression levels of the top 30 genes. Observe that the expression values have a different range and heatmap signature for each of the six different preprocessing methods. This will have an impact in the graph that is created from the preprocessed data.

all pairs of transcript probes. Table 1.1 lists a collection of similarity metrics. This dissertation focuses on a select few metrics that are popular in the bioinformatics community [60], including Pearson product-moment, Spearman Rank, Kendall's tau, Euclidean distance, and mutual information.

3.4.1 Pearson Product-moment

One of the most popular metrics used to determine similarity in microarray analysis is the Pearson product-moment correlation coefficient, see Equation 3.1. This correlation produces a measure of linear dependence between random variables, X and Y, in the range [-1,1]. A positive correlation occurs when variable X increases, then variable Y increases. If variable X increases and variable Y decreases, then a negative correlation is produced. A Pearson's correlation of 1 is a perfect positive correlation, and -1 being a perfect negative correlation. Perfect correlations occur when the plotting the variable X against variable Y creates a straight line. A correlation of 0 is interpreted to mean that given the value of variable X, the value of variable Y is unknown. Pearson's correlation assumes that the data is normally distributed and it is very sensitive to outliers. Figure 3.3 illustrates the relationship of a Pearson correlation between random variables.

$$\text{Pearson Correlation} = \frac{\sum(X_i - \overline{X})(Y_i - \overline{Y})}{(n-1)S_x S_y} \tag{3.1}$$

3.4.2 Spearman's Rank and Kendall's Tau

Spearman's rank correlation coefficient and Kendall's tau rank correlation coefficient are non-parametric measures of dependence between two random variables, X and Y. Both of these metrics use the rank of an observation, within the spectrum of a variable, to determine dependence. Table 3.2 illustrates the ranking function as applied to two genes. Spearman's correlation can be calculated using the formula for Pearson's correlation, given that if a tie occurs in the ranking, the average of the

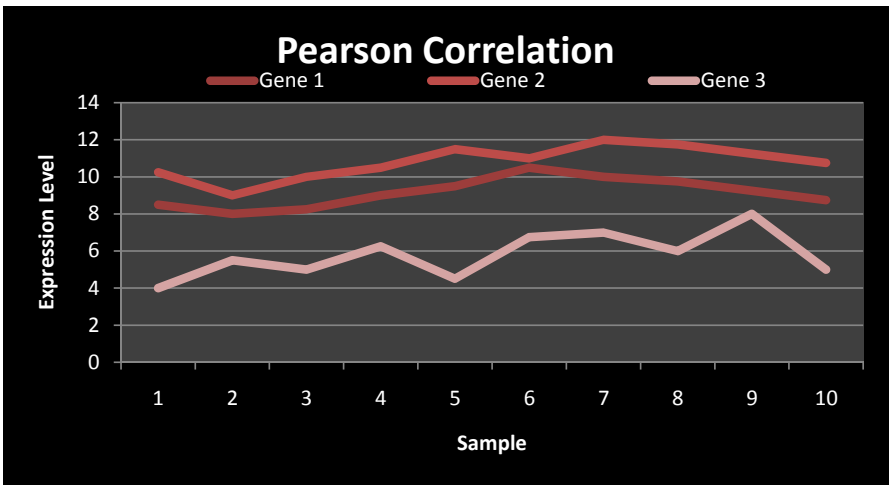


Figure 3.3: Example of Pearson correlation given 3 genes and 10 samples. The Pearson correlation between Gene 1 and Gene 2 is 0.82. Observe that, generally speaking, when the expression level of Gene 1 rises, the expression level of Gene 2 also rises, and when the expression level Gene 1 falls, so does the expression level of Gene 2. The Pearson correlation between Gene 1 and Gene 3 is 0.53 and the Pearson correlation of Gene 2 and Gene 3 is 0.39. The low correlation value between Gene 2 and Gene 3 can be attributed to the expression levels of Gene 2 rising, when over the same samples, the expression levels of Gene 3 are falling, and vice versa.

ranking is used. If no ties occur, Spearman's correlation can be calculated using the formula in Equation 3.2. Kendall's tau rank correlation coefficient, Equation 3.3, uses the number of paired concordant and discordant observations to assign dependence. Both Spearman and Kendall correlations fall in the range of [-1,1].

$$\text{Spearman Correlation} = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \tag{3.2}$$

$$\text{Kendall Correlation} = \frac{n_c - n_d}{(1/2)n(n-1)} \tag{3.3}$$

3.4.3 Euclidean Distance

A popular distance metric is Euclidean distance, as calculated by the Pythagorean formula. This metric measures the distance between a pair of variables in n-dimensional space, where n is the number of observations, as seen in Equation 3.4.

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Gene 1	8.5	8	8.25	9	9.5	10.5	10	9.75	9.25	8.75
Rank	3	1	2	5	7	10	9	8	6	4
Gene 2	10.25	9	10	10.5	11.5	11	12	11.75	11.25	10.75
Rank	3	1	2	4	8	6	10	9	7	5

Table 3.2: Spearman rank correlation converts the expression levels into the respective rank of the expression level within the given gene. For example, the expression level of Gene 1 in sample 1 is 8.5. This is the 3rd smallest expression value within Gene 1 (across all samples), so it is assigned a rank of 3. The Spearman correlation between these two genes is 0.86, as compared to the Pearson correlation value of 0.82.

$$\text{Euclidean Distance} = \sqrt{\sum (X_i - Y_i)^2} \tag{3.4}$$

Euclidean distance follows all the rules set forth in [61] to be a valid distance metric and is often used as the default metric for generating hierarchical clustering dendrograms, as illustrated in Figure 3.4.

3.4.4 Mutual Information

Mutual information is a measure of mutual dependence between two independent variables, X and Y, that, unlike the previous methods, is not limited to measuring only linear dependence. Mutual information produces nonnegative correlations, and is gaining acceptance as a similarity metric in biological analysis [62]. Equation 3.5 gives the formula for Mutual information as presented in [63].

$$\text{Mutual Information} = \sum_y \sum_x p(x,y) log(\frac{p(x,y)}{p1(x)p2(y)))} \tag{3.5}$$

A drawback to mutual information the requirement that the observations be placed into n bins, where n is defined by the user. The number of bins chosen has an effect

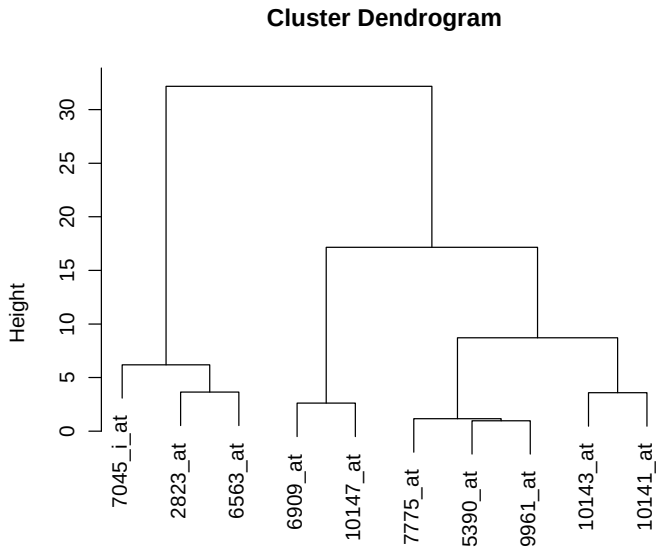


Figure 3.4: Euclidean distance is the default metric for the hclust method in R. This dendrogram uses the Euclidean distance value to cluster the first 10 genes from the Yeast dataset. Selecting a value on the vertical axis determines how many groups, or clusters, are defined at that threshold. For example, at height 10, there are 3 clusters.

on the final coefficient value produced by the algorithm. Table 3.3 lists the different mutual information coefficients between the same two variables, given that the data is broken into a different number of bins.

Number of bins	6	7	8	9	10	11	12	13	14	15
Mutual Information Gene 1 vs Gene2	1.14	1.33	1.6	1.74	1.88	2.16	2.16	2.3	2.3	2.3

Table 3.3: The similarity measure produced by mutual information is dependent on the parameter selected for bin size. By default, the bin size is 10. However, as illustrated above, the bin size can have a large influence in the similarity value produced.

3.4.5 Results

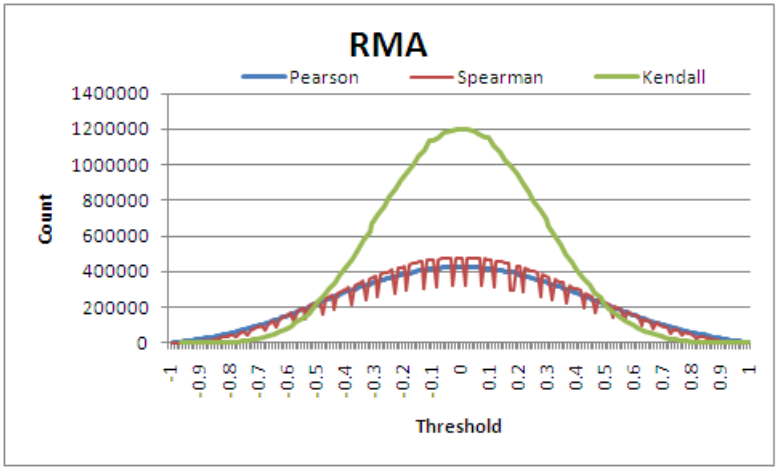
The results from the six different preprocessing methods are used as input to each of the above mentioned similarity metrics. The Pearson, Spearman, and Kendall

correlations are all contained within the range [1,1] and are plotted on the same graph. Both the Euclidean distance and Mutual information measures have a lower bound of 0, but they do not have an upper bound and as such, they are plotted on separate graphs. For all pairwise computations, at least 10 common observations are required in order to generate the respective measurement.

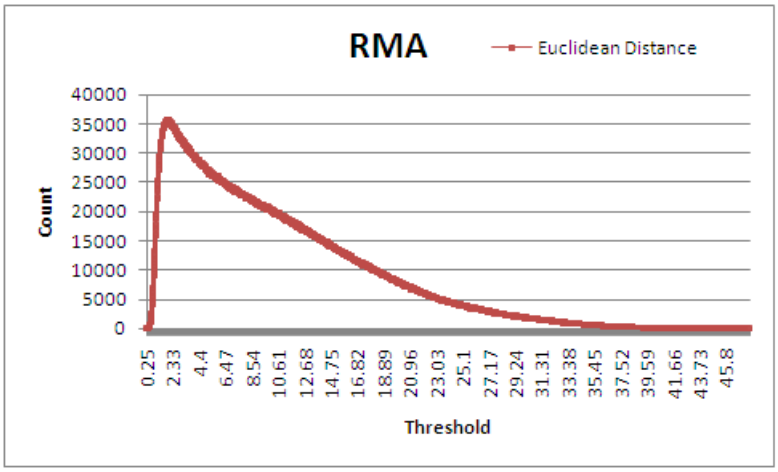
The distribution of the similarity values using the RMA preprocessing method is examined in Figure 3.5. In Figure 3.5(a), the distribution of the values for the Kendall rank correlation has a large spike centered around 0 and has shorter tails, whereas both the RMA and Spearman correlations have longer tails. The longer tails are a product of the number of high correlations produced, and the genes of interest lies within these outermost part of these tails. The Euclidean distance measure, as applied to the RMA method, shows a spike in similarity values around 2 and slowly trails off. The genes with the smallest Euclidean distance has the highest similarity value, therefore, the genes of interest lie close to 0. Finally, the Mutual Information measure is seen in Figure 3.5(c). The similarity values produced fall within the range of 0 to around 2. The similarity values that are high represent the genes of interest.

A similar distribution pattern can be seen for the remaining preprocessing methods: MAS 5.0, GCRMA, GCRMA MM, DCHIP, DCHIP MM, see Figures 3.6 to 3.10. Even though the distributions are similar, there are some slight variations. For example, the Kendall rank correlation has a higher peak when combined with certain preprocessing methods, namely MAS 5.0 and GCRMA. The distribution of the Kendall rank correlation when used in conjunction with the DCHIP MM preprocessing has the smallest peak at the correlation value of 0, with ∼100,000 correlations, and also has small perturbations in the distribution of the correlation values. This is a direct impact of the background subtraction method used in the DCHIP MM algorithm and the requirement that at least 10 observations must be in common between a the pair of genes/probes being measured.

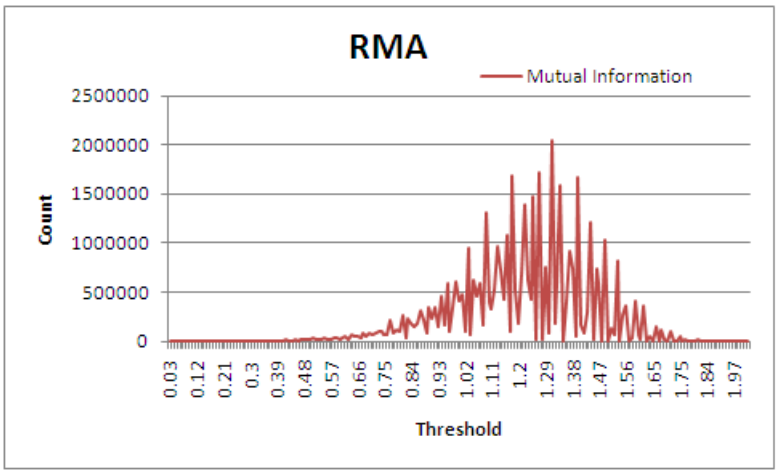
The distributions of the Pearson correlation values, derived from each of the six preprocessing methods, are seen in Figure 3.11(a). Recall that Pearson correlation



(a)

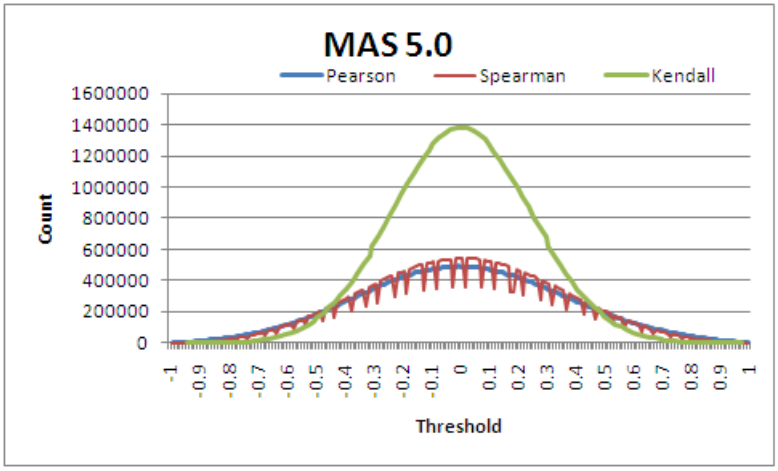


(b)

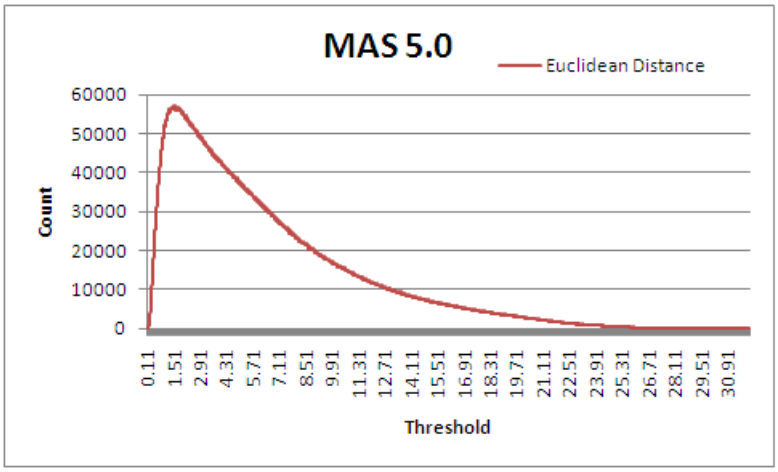


(c)

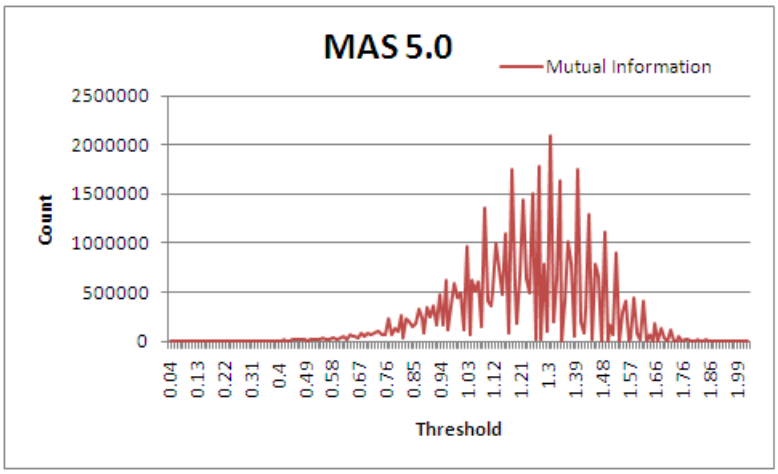
Figure 3.5: Distribution of the RMA preprocessed data with (a) Pearson, Spearman, and Kendall correlations (b) Euclidean distance, and (c) Mutual Information.



(a)

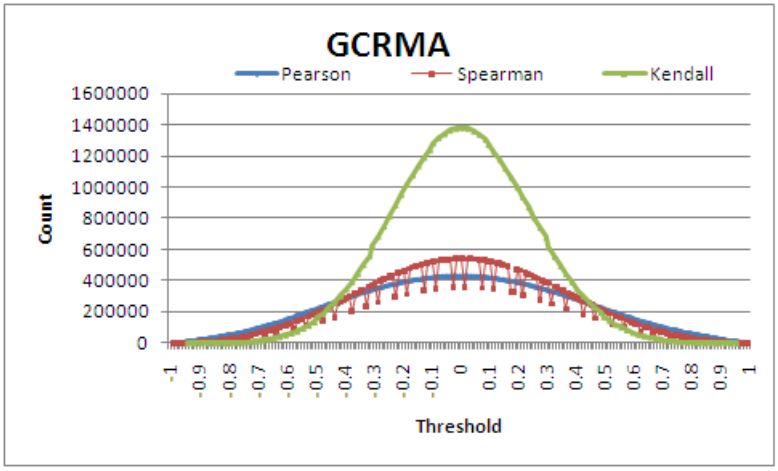


(b)

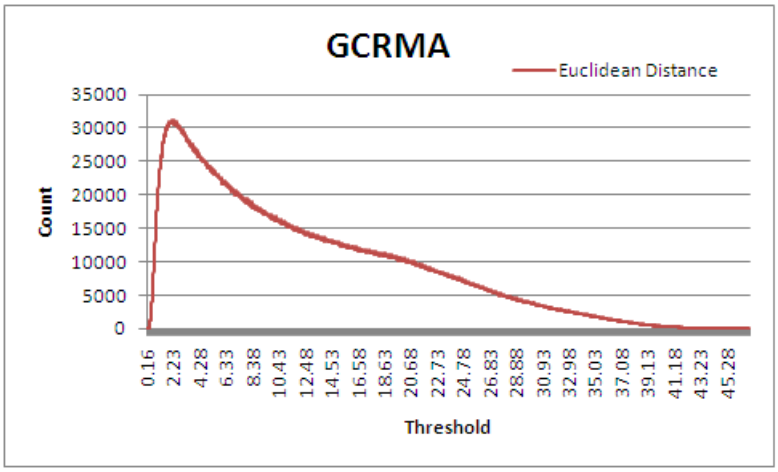


(c)

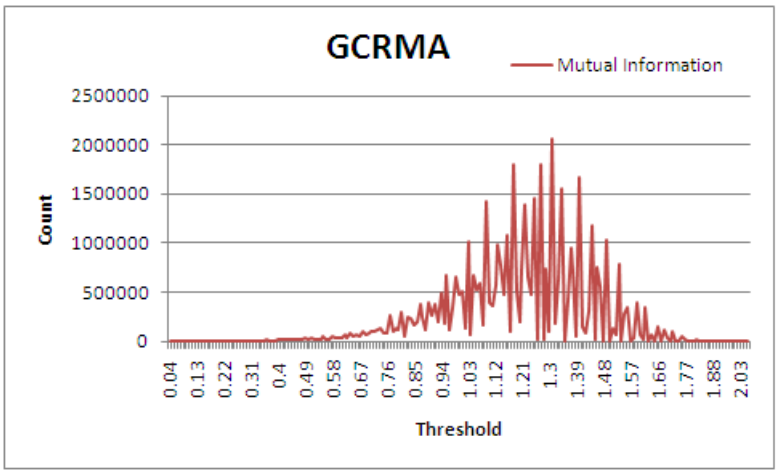
Figure 3.6: Distribution of the MAS 5.0 preprocessed data with (a) Pearson, Spearman, and Kendall correlations (b) Euclidean distance, and (c) Mutual Information.



(a)

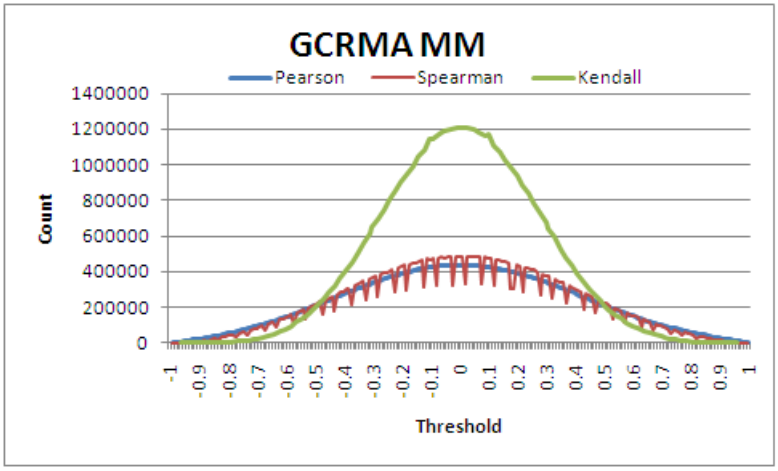


(b)

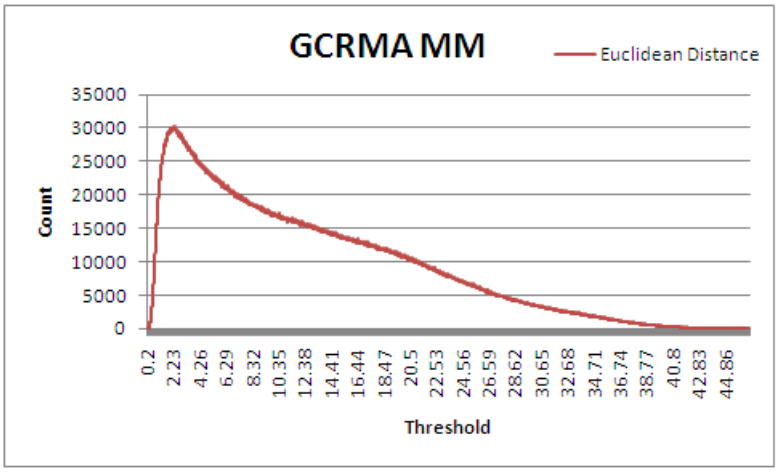


(c)

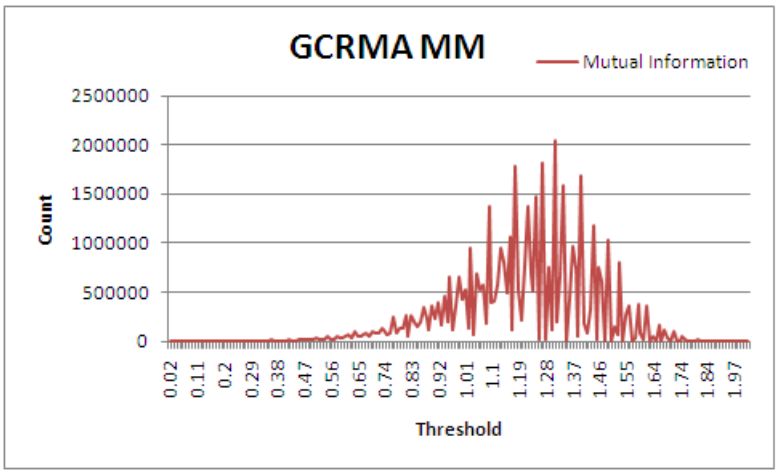
Figure 3.7: Distribution of the GCRMA preprocessed data with (a) Pearson, Spearman, and Kendall correlations (b) Euclidean distance, and (c) Mutual Information.



(a)

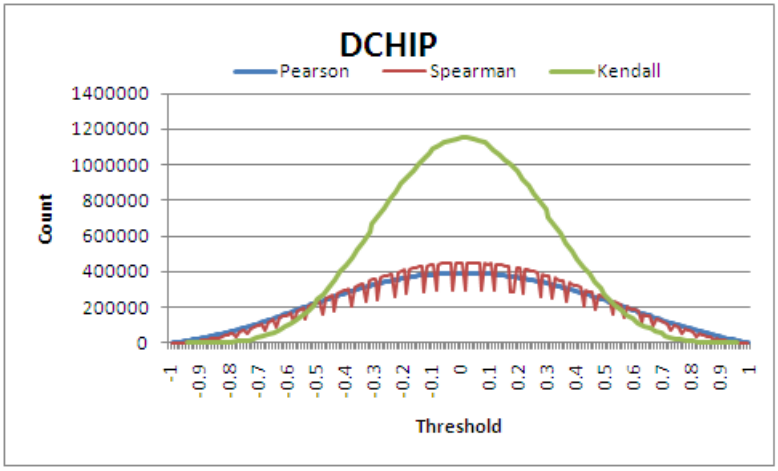


(b)

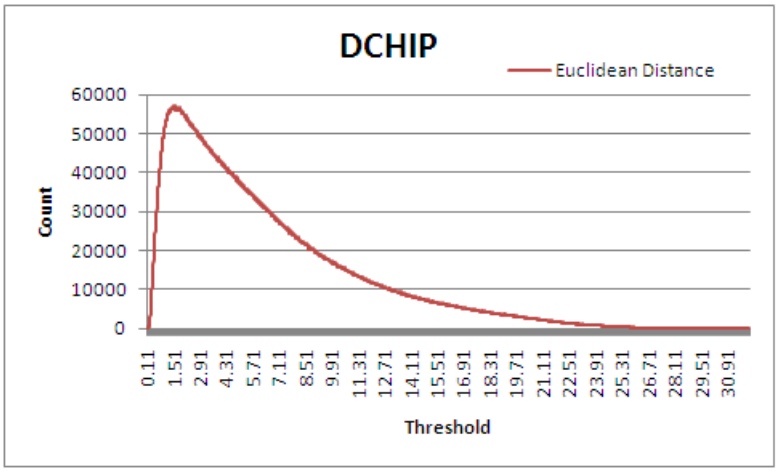


(c)

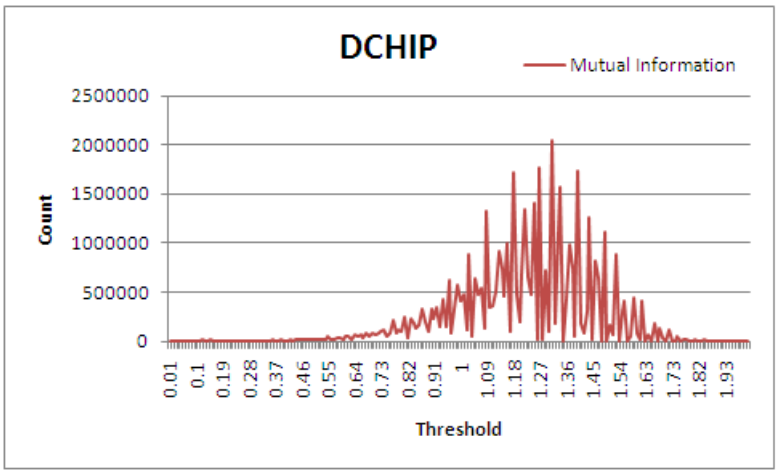
Figure 3.8: Distribution of the GCRMA MM preprocessed data with (a) Pearson, Spearman, and Kendall correlations (b) Euclidean distance, and (c) Mutual Information.



(a)

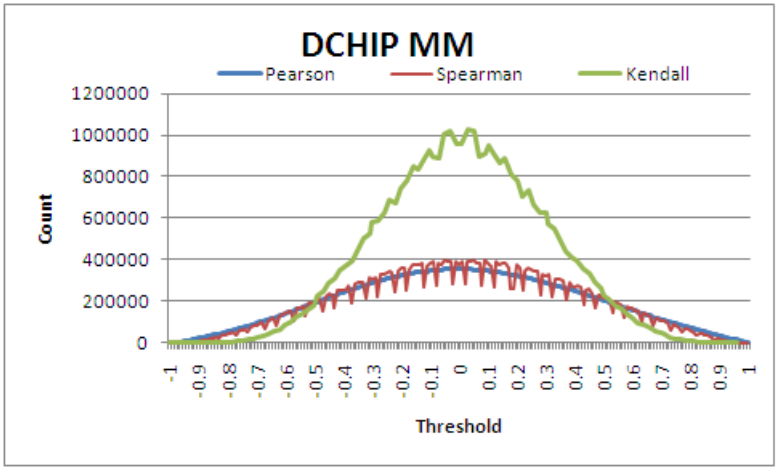


(b)

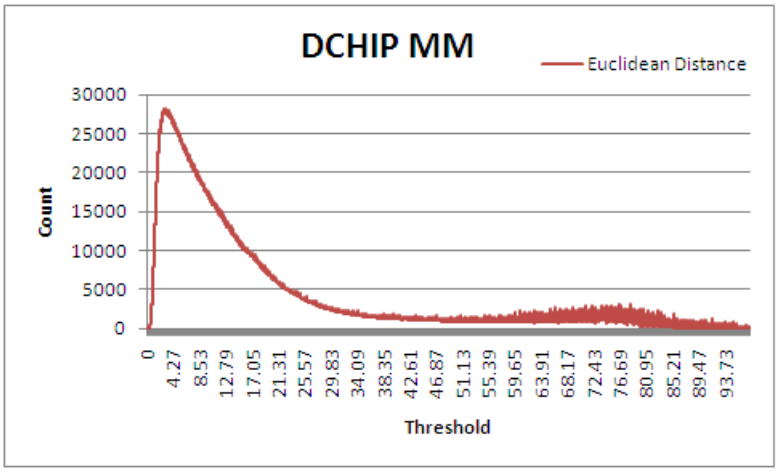


(c)

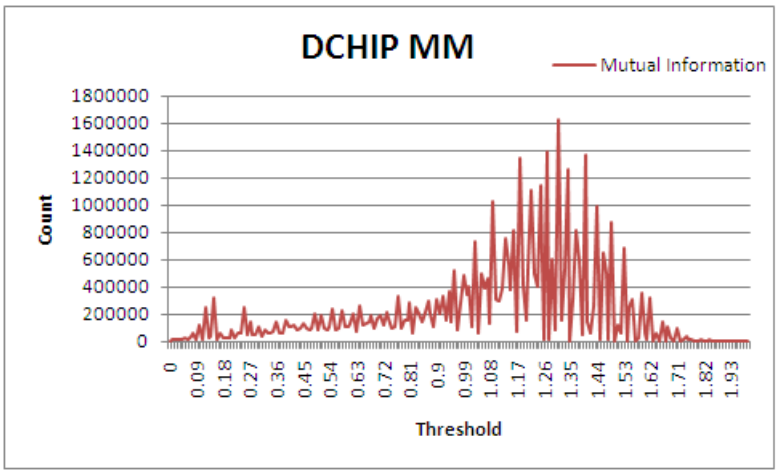
Figure 3.9: Distribution of the DCHIP preprocessed data with (a) Pearson, Spearman, and Kendall correlations (b) Euclidean distance, and (c) Mutual Information.



(a)



(b)



(c)

Figure 3.10: Distribution of the DCHIP MM preprocessed data with (a) Pearson, Spearman, and Kendall correlations (b) Euclidean distance, and (c) Mutual Information.

values close to 0 signify a pair of uncorrelated genes, and that it is expected to be normally distributed around 0. The density of the graphs produced using threshold values between 0.75 and 0.99 are seen in Figure 3.11(b). Recall that the density of a graph is defined as

$$\mathrm{D} = \frac{2|E|}{|V|(|V|-1)}$$

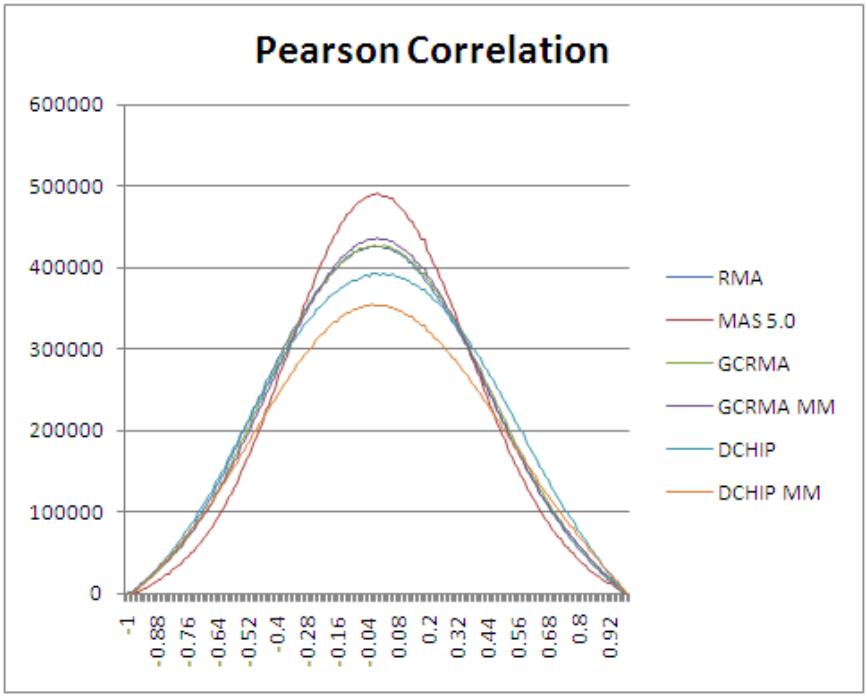
where $0 \leq D \leq 1$. Starting with the correlation threshold of 0.99 and lowering the threshold by 0.01 at each interval, the density tends to increase as the threshold decreases. A contradiction to this occurs at the highest thresholds where the graph consists of only a few vertices and edges, thus a spike is seen at the far right of the plot. After the threshold is lowered and more vertices are included in the graph, the density of the graph then follows the pattern mentioned above. Table 3.4 lists a sample of the number of vertices, number of edges and the density for each of the 26 threshold values from the RMA preprocessed data. Similar results for the remaining five similarity metrics are presented in Figures 3.12 to 3.15.

THRESHOLD LEVEL	0.75	0.80	0.85	0.90	0.95	0.99
Number of Vertices	9308	9010	7858	5251	2298	202
Number of Edges	1628048	973129	507392	202317	38648	259
Density	0.075	0.048	0.033	0.029	0.029	0.026

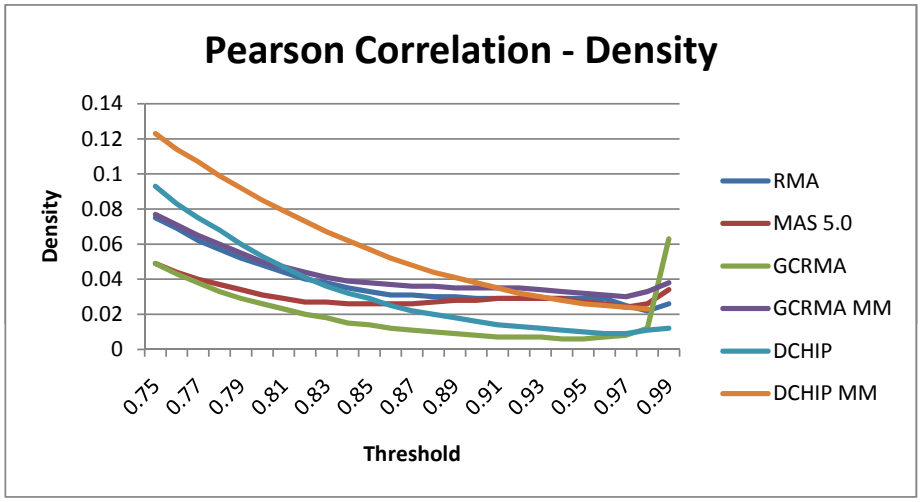
Table 3.4: Graphs were created from data that were preprocessed using the RMA algorithm and the similarity measures were generated using Pearson correlations. Twenty-six different threshold levels [0.75,0.99] were used to generate 26 different graphs. Graph metrics listed above were extracted from 6 of the 26 graphs.

3.5 Multiple testing correction

The values produced by the similarity metrics mentioned above can be submitted to a significance test. The significance level, denoted by α, is used to determine if the

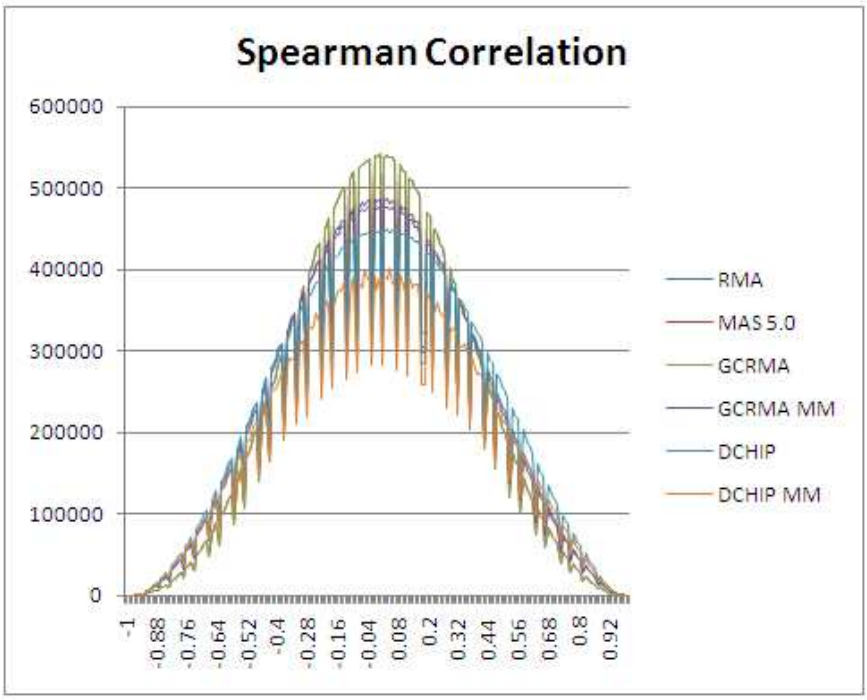


(a)

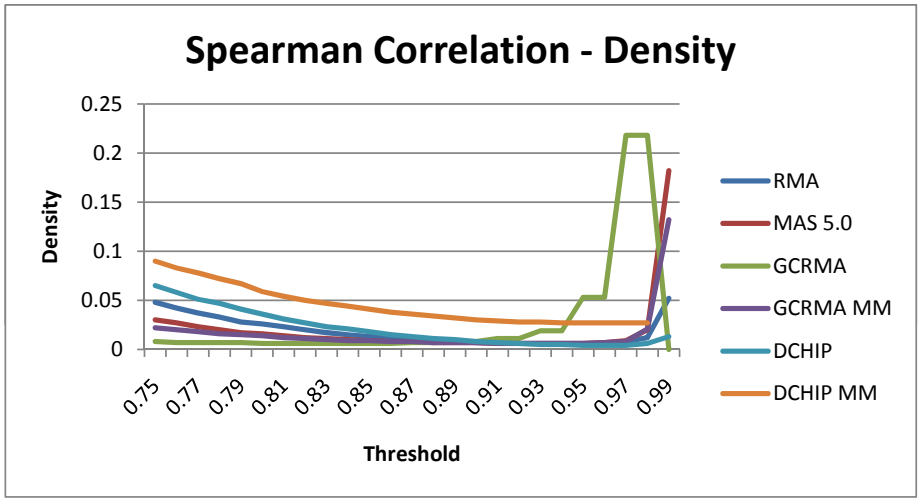


(b)

Figure 3.11: Pearson correlation values generated using the six different preprocessing methods RMA, MAS 5.0, GCRMA, GCRMA MM, DCHIP, DCHIP MM.

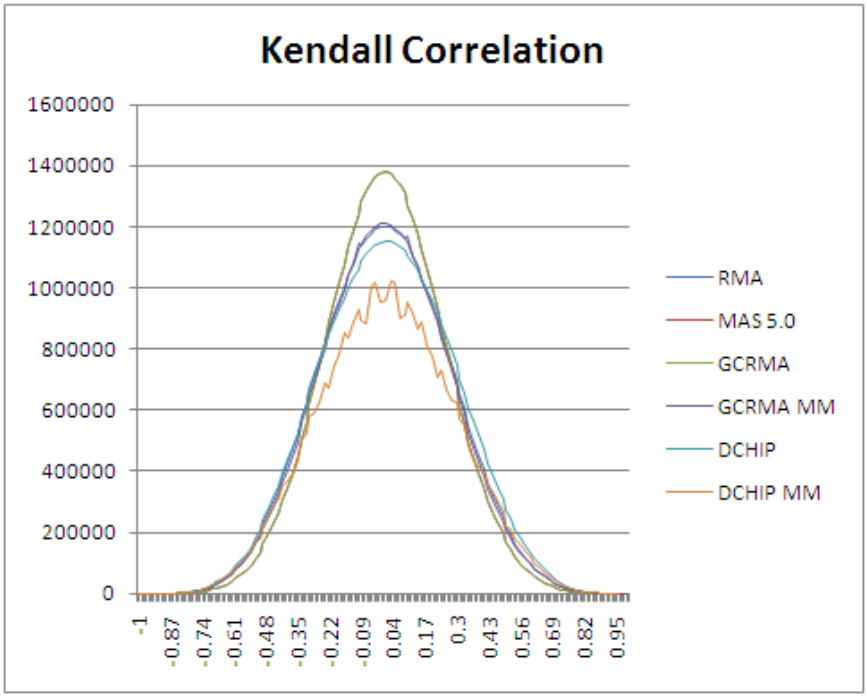


(a)

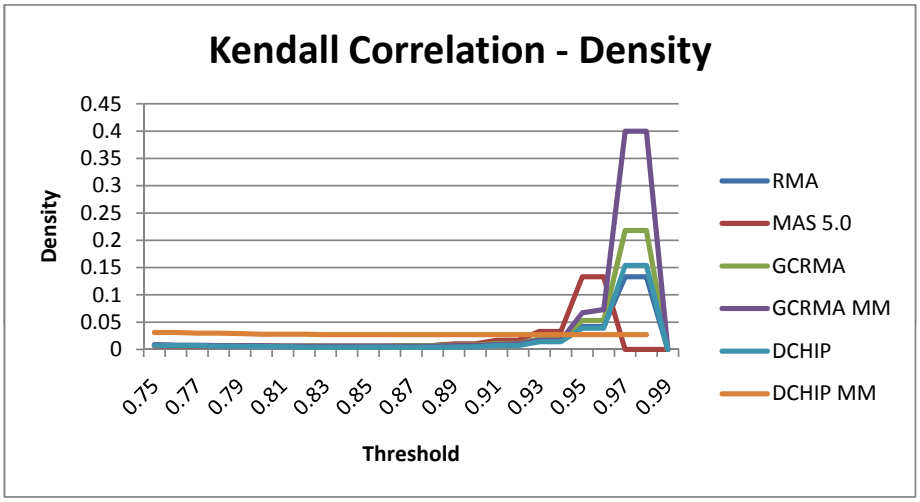


(b)

Figure 3.12: Spearman correlation values generated using the six different preprocessing methods RMA, MAS 5.0, GCRMA, GCRMA MM, DCHIP, DCHIP MM.

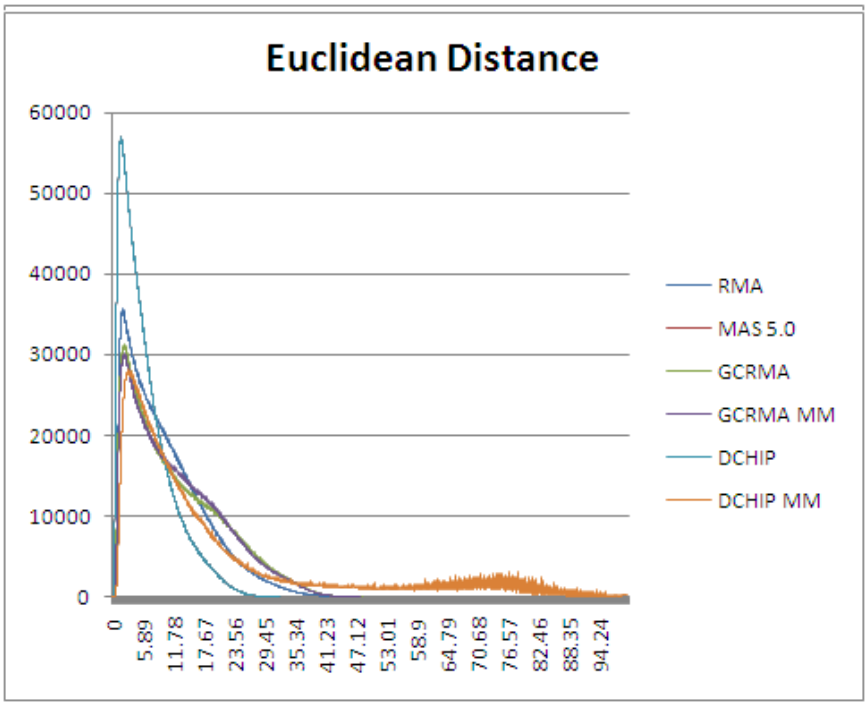


(a)

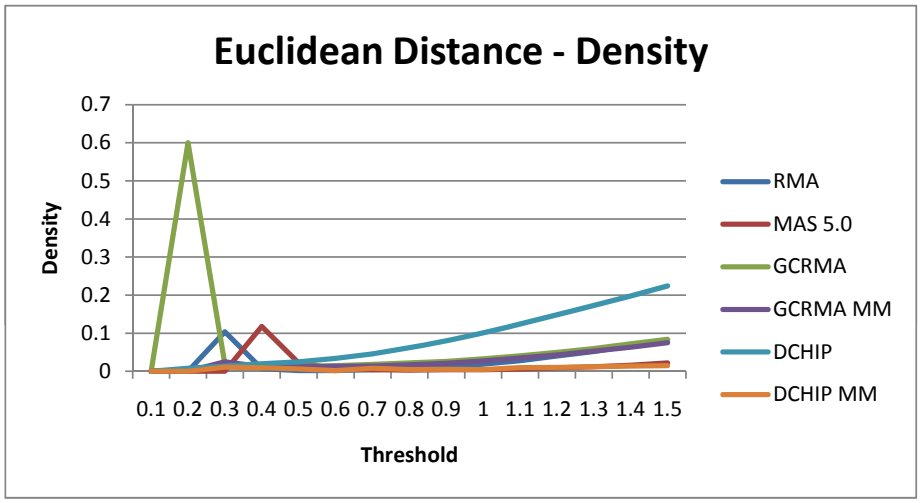


(b)

Figure 3.13: Kendall correlation values generated using the six different preprocessing methods RMA, MAS 5.0, GCRMA, GCRMA MM, DCHIP, DCHIP MM.

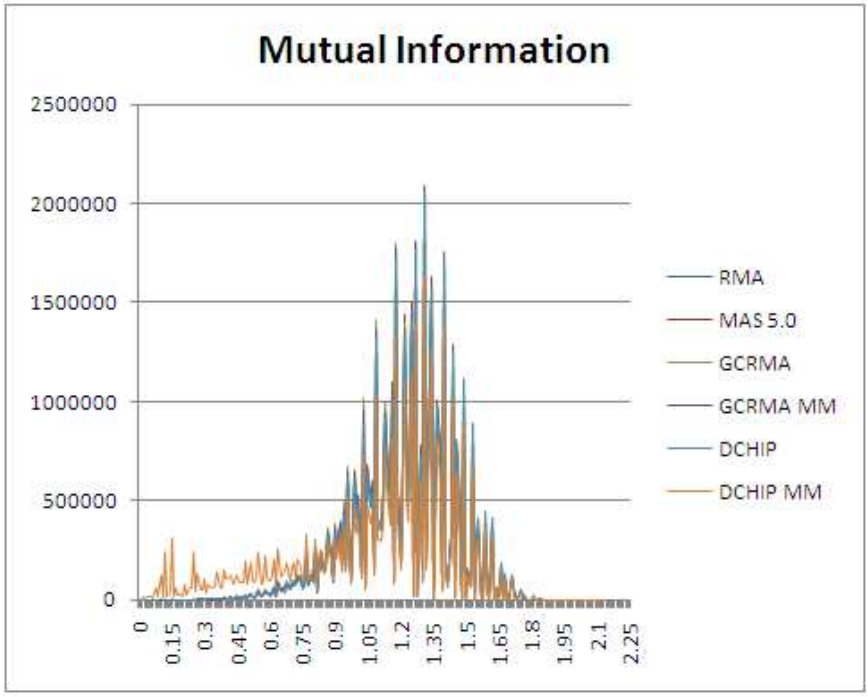


(a)

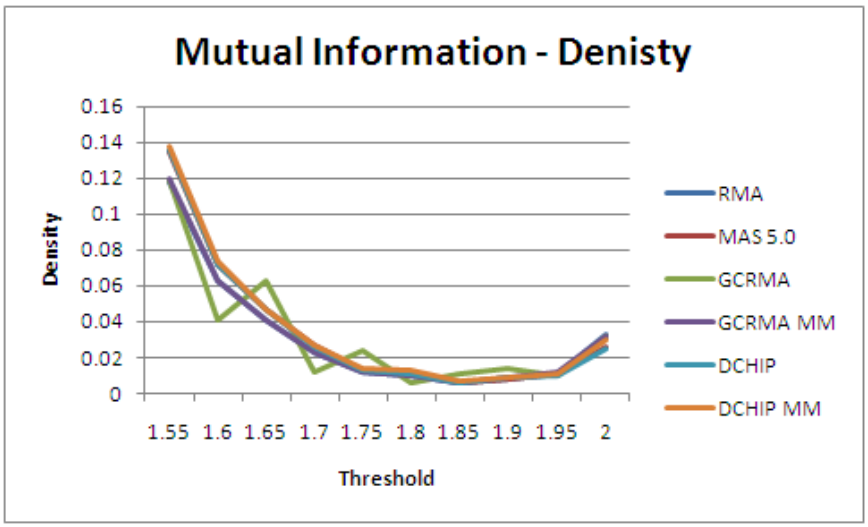


(b)

Figure 3.14: Euclidean distance values generated using the six different preprocessing methods RMA, MAS 5.0, GCRMA, GCRMA MM, DCHIP, DCHIP MM.



(a)



(b)

Figure 3.15: Mutual information values generated using the six different preprocessing methods RMA, MAS 5.0, GCRMA, GCRMA MM, DCHIP, DCHIP MM.

null hypothesis, H_0 is accepted or rejected. The p-value of a test is defined to be the probability of observing the data at least as extreme as that observed, given that the null hypothesis is true. If a p-value is lower than α, then H_0 is rejected and the alternative, H_a is accepted. In other words, the p-value is a metric used to determine whether or not an observation occurred by chance. The levels of α are frequently set to 0.1, 0.05, 0.01, or 0.001. While the levels of α are arbitrary, $\alpha = 0.05$ is generally accepted as the de facto standard. The level of α is sometimes referred to as the minimum false positive rate, as it is the rate that false positives are expected to occur. An example of the false positive rate would be testing differential expression between 20,000 genes. Given $\alpha = 0.05$, then it is expected that 5%, or 1,000 genes, will be false positives. That is they are, in fact, not differentially expressed in reality, but given their p-values, they are classified as being differentially expressed. An equation for the false positive rate is given in Equation 3.6 [64].

$$\text{false positive rate} \approx \frac{\text{number of false positives}}{\text{number of true null tests}} \tag{3.6}$$

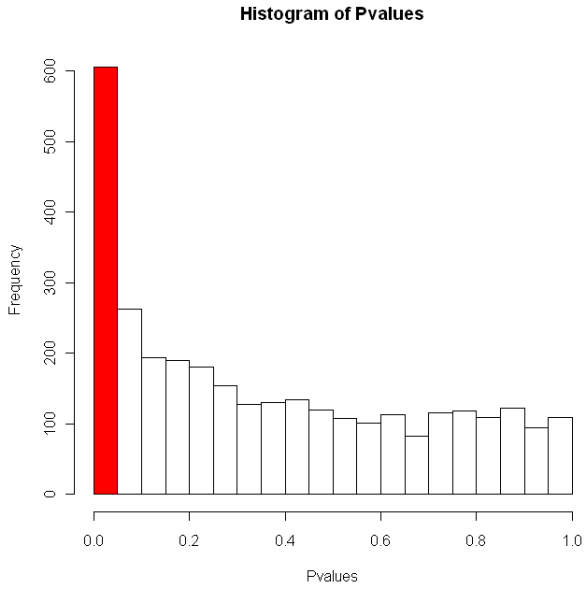
The Student's t-test is one test of significance that given a Pearson correlation value, produces a p-value based on the degrees of freedom. Applying the Student's t-test, presented in Equation 3.7, to microarray analysis relies on the assumption that the correlations generated by the microarray data is normally distributed. If that is not the case, then another method for generating p-values is permutation testing. A permutation test is a test of significance that produces the distribution of the test statistic by permuting the labels of the observed data and generating all possible values for said test statistic. This distribution is then used to compare the observed test statistic against the α level to determine significance. Permutation tests can be used for any test statistic, but they are generally used only when the distribution is invalid or unknown. A drawback to using permutation tests is they are computationally expensive.

$$t = \frac{\overline{X}_1 - \overline{Y}_1}{S_{x_1 y_1} * \sqrt{\frac{2}{n}}} \tag{3.7}$$

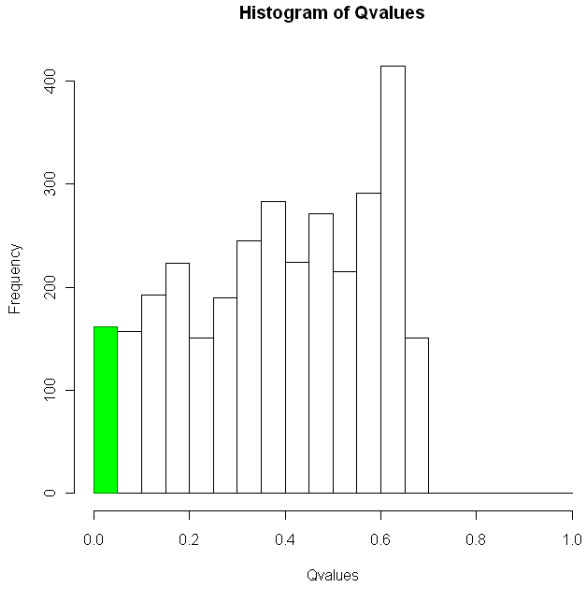
The question of significance must also take into account multiple testing. Methods for correcting multiple tests include Bonferroni and FDR, as discussed in [65]. The formula for Bonferroni was presented in Equation 1.1 and is, computationally, the easiest method to generate an *adjusted p-value*. However, the tradeoff is that Bonferroni is typically too conservative in its correction method, reducing both the number of false positives and true discoveries. FDR, presented in Equation 3.8, is another popular method for multiple testing correction in microarray analysis [66]. FDR introduces a method to control the number of false positives by setting an acceptable cutoff level, typically 5%, that is restricted only to the tests deemed significant based on their p-values.

$$\text{false discovery rate} \approx \frac{\text{number of false positives}}{\text{number of significant tests}} \tag{3.8}$$

It was mentioned previously that p-values are generated based on the null distribution that is known, such as the t distribution when the data is normal [67], or the null distribution is generated using permutation tests. The computation of p-values relies solely on the distribution and is therefore easy to compute. FDR, however, is not dependent on the null distribution, and the computations are more complex. Storey showed in [64] that the q-values produced by FDR can be successfully computed from the distribution of all the p-values at once, or computed directly from the original data. Q-values are analogous to p-values in the sense that they are the threshold used to define significance for a test. Like p-values, typical q-value levels are 0.05 or 0.01. An example of the false discovery rate would be identifying a set of 1,000 genes to be differentially expressed. Using a q-value threshold of 0.05, 50 of these *discovered* genes are expected to be false positive. Figure 3.16 illustrates the relationship between p-values and q-values.



(a)



(b)

Figure 3.16: The QVALUE package, along with the Hedenfalk dataset [68], was used to generate an example of the differences in the set of genes identified as significant using only the (a)pvalue and (b) qvalue.

3.6 Threshold Selection

The third and final category in creating a graph is threshold selection. Up to this point, a complete graph can be generated using any combination of the above methods. To review the graph creation process, the vertices of the graph are the genes extracted from the data and all pairs of vertices have a weighted edge, representing the level of interaction, connecting them. The weight of the edge comes directly from the combination of the preprocessing algorithm and similarity metric. In order to reduce the graph, retaining only those genes and interactions that are putatively biologically significant, a threshold must be applied to the graph. Many threshold selection algorithms have been proposed and studied [37, 38]. These algorithms range from simple statistical oriented thresholds, such as p-values and q-values, to graph based properties, such as the number of maximal cliques in a graph. This dissertation will incorporate some of the proposed threshold selection methods in [37] to illustrate the effects of threshold selection on the resulting graph. These methods include:

- **Significant p-values** - Retain edges such that $p - value <= threshold$, where $threshold \in \{0.1, 0.05, 0.01\}$.
- **Significant q-values** - Retain edges such that $q - value <= threshold$, where $threshold \in \{0.1, 0.05, 0.01\}$.
- **Maximal Clique-2** - Set *threshold* to the value at which the number of maximal cliques double from the previous level, and keep any edge with a value greater than the threshold.
- **Maximal Clique-3** - Set *threshold* to the value at which the number of maximal cliques triple from the previous level, and keep any edge with a value greater than the threshold.
- **Top 1%** - Keep edges with similarity metric value in the top 1% of all possible values.

- **Top 0.1%** - Keep edges with similarity metric value in the top 0.1% of all possible values.

The threshold selection methods were applied to the same Yeast data, using each possible combination of preprocessing method and similarity metric. Recall that there are six different preprocessing methods, five different similarity metrics, and six different threshold selection methods. All possible combinations leads to the generation of 180 graphs. The respective threshold levels for each of these 180 graphs are listed in Tables 3.5 to 3.9. P-values and q-values were both set to 0.05. Though the selection of 0.05 is an arbitrary value, it is widely accepted in practice as the de facto standard. The maximal-clique 2 and maximal-clique 3 thresholds were selected as described in [37], and the thresholds for the top 1% and top 0.1% were selected using the absolute value of all correlations and the nearest threshold value that contained the 99th percentile of the correlations was chosen.

The threshold values produced by a given algorithm is heavily dependent on the preprocessing method and the similarity metric used on the data. Although similar threshold values are generated, see Maximal-Clique 2 in Table 3.5, the graphs that are generated from these thresholds vary widely. It is important to note that even if the same threshold values were produced, that the graphs generated would still differ in metrics such as number of vertices, number of edges edges, density, etc. This is due to the fact that the similarity metric and/or preprocessing step was different.

Pearson Correlation						
	RMA	MAS 5.0	GCRMA	GCRMA MM	DCHIP	DCHIP MM
p-value	0.05	0.05	0.05	0.05	0.05	0.05
q-value	0.05	0.05	0.05	0.05	0.05	0.05
maximal clique-2	0.95	0.94	0.95	0.95	0.92	0.9
maximal clique-3	0.94	0.93	0.93	0.94	0.9	0.89
top 1%	0.91	0.89	0.91	0.91	0.91	0.9
top 0.1%	0.95	0.94	0.94	0.95	0.95	0.94

Table 3.5: Pearson correlation threshold levels generated using the five different threshold selection methods p-value, q-value, maximal-clique 2, maximal-clique 3, top 1%, and top 0.1%

Spearman Correlation						
	RMA	MAS 5.0	GCRMA	GCRMA MM	DCHIP	DCHIP MM
p-value	0.05	0.05	0.05	0.05	0.05	0.05
q-value	0.05	0.05	0.05	0.05	0.05	0.05
maximal clique-2	0.88	0.87	0.89	0.88	0.91	0.9
maximal clique-3	0.87	0.86	0.88	0.86	0.89	0.88
top 1%	0.87	0.85	0.87	0.84	0.88	0.89
top 0.1%	0.92	0.91	0.93	0.91	0.93	0.93

Table 3.6: Spearman rank threshold levels generated using the five different threshold selection methods p-value, q-value, maximal-clique 2, maximal-clique 3, top 1%, and top 0.1%

Kendall Correlation						
	RMA	MAS 5.0	GCRMA	GCRMA MM	DCHIP	DCHIP MM
p-value	0.05	0.05	0.05	0.05	0.05	0.05
q-value	0.05	0.05	0.05	0.05	0.05	0.05
maximal clique-2	0.79	0.77	0.76	0.76	0.76	0.75
maximal clique-3	0.76	0.75	0.74	0.75	0.74	0.74
top 1%	0.81	0.79	0.81	0.75	0.82	0.82
top 0.1%	0.88	0.86	0.88	0.81	0.88	0.89

Table 3.7: Kendall tau threshold levels generated using the five different threshold selection methods p-value, q-value, maximal-clique 2, maximal-clique 3, top 1%, and top 0.1%

Euclidean Distance						
	RMA	MAS 5.0	GCRMA	GCRMA MM	DCHIP	DCHIP MM
p-value	0.05	0.05	0.05	0.05	0.05	0.05
q-value	0.05	0.05	0.05	0.05	0.05	0.05
maximal clique-2	0.7	1.2	0.7	0.7	0.4	0.4
maximal clique-3	0.9	1.4	0.8	0.8	0.5	0.6
top 1%	1.04	0.58	0.97	1.02	0.58	1.38
top 0.1%	0.72	0.35	0.61	0.65	0.35	0.86

Table 3.8: Euclidean distance threshold levels generated using the five different threshold selection methods p-value, q-value, maximal-clique 2, maximal-clique 3, top 1%, and top 0.1%

Mutual Information						
	RMA	MAS 5.0	GCRMA	GCRMA MM	DCHIP	DCHIP MM
p-value	0.05	0.05	0.05	0.05	0.05	0.05
q-value	0.05	0.05	0.05	0.05	0.05	0.05
maximal clique-2	1.65	1.65	1.65	1.65	1.65	1.65
maximal clique-3	1.7	1.7	1.7	1.7	1.7	1.7
top 1%	1.65	1.65	1.65	1.65	1.65	1.65
top 0.1%	1.75	1.75	1.75	1.75	1.8	1.8

Table 3.9: Mutual information threshold levels generated using the five different threshold selection methods p-value, q-value, maximal-clique 2, maximal-clique 3, top 1%, and top 0.1%

3.7 Graph Creation and Results

Using all possible combinations of the algorithms in the three steps to create a graph, a total of 180 different graphs were generated from the Yeast data. The graph properties such as number of vertices, number of edges, density, and number of connected components are used to discuss the effectiveness of each threshold selection method. The 30 graphs derived from each of the different threshold selection methods are presented in Table 3.10 to 3.15.

Restricting focus to the Pearson correlation column in Table 3.12, it can be observed that the graphs produced vary widely with the number of vertices and edges ranging from [2686,3038] and [56132,64112], respectively. The densities of these graphs range from [0.0139,0.016], while the number of connected components fall in the range of [127, 137].

Comparing the size of the resulting graphs, the p-value threshold selection method produced the lowest thresholds and thus generated graphs with the largest number of vertices and edges, see Table 3.10. Although the graphs created using this threshold levels are statistically significant networks, the graphs are typically very large and true biological interactions are lost due to the noise in this data. Similar to the p-value, the q-value threshold selection method method tries to reduce the number of false-discoveries, however, the graphs generated using this technique also produces very large, dense graphs, see Table 3.11. The methods that select the top 1% or top 0.1% of all similarity values tends to produce high threshold values, but lack the ability to adjust threshold levels in response to different different edge distributions in the graph, see [38]. The Maximal-Clique based techniques produce thresholds that generate graphs based not only on statistically significant edges, but on inherent graph structures in the data.

The preprocessing methods, similarity metrics, and threshold selection tools all have an effect in the final outcome of the generation of a graph. It is not the case that a single combination of the three metrics mentioned above will always give the

best results. It is advisable that all possible combinations of the three different steps should be attempted whenever possible. This allows the end user to determine which combination of the methods will produce the graph best suited for their analysis. As it applies to this particular data, one advisable metric combination would be RMA normalization, Pearson correlation, and Maximal clique-2 threshold selection. On average, this metric combination ranks as one of the best when taking into account graph properties such as size and density.

Pvalue 0.05	Pearson Correlation	Spearman Correlation	Kendall Correlation	Euclidean Correlation	Mutual Information
RMA	Vertices: 9335 Edges: 8249258 Density: 0.1893 Components: 1	Vertices: 9335 Edges: 7234215 Density: 0.1661 Components: 1	Vertices: 9335 Edges: 8234215 Density: 0.189 Components: 1	Vertices: 9335 Edges: 7923412 Density: 0.1819 Components: 1	Vertices: 9335 Edges: 7823522 Density: 0.1796 Components: 1
MAS 5.0	Vertices: 9335 Edges: 8890151 Density: 0.2041 Components: 1	Vertices: 9335 Edges: 6819645 Density: 0.1565 Components: 1	Vertices: 9335 Edges: 8571245 Density: 0.1967 Components: 1	Vertices: 9335 Edges: 7733705 Density: 0.1775 Components: 1	Vertices: 9335 Edges: 8121914 Density: 0.1864 Components: 1
GCRMA	Vertices: 9335 Edges: 7752765 Density: 0.178 Components: 1	Vertices: 9335 Edges: 6912773 Density: 0.1587 Components: 1	Vertices: 9335 Edges: 8606557 Density: 0.1976 Components: 1	Vertices: 9335 Edges: 8539663 Density: 0.196 Components: 1	Vertices: 9335 Edges: 7418057 Density: 0.1703 Components: 1
GCRMA MM	Vertices: 9335 Edges: 8851768 Density: 0.2032 Components: 1	Vertices: 9335 Edges: 6926624 Density: 0.159 Components: 1	Vertices: 9335 Edges: 7750192 Density: 0.1779 Components: 1	Vertices: 9335 Edges: 7646948 Density: 0.1755 Components: 1	Vertices: 9335 Edges: 7271207 Density: 0.1669 Components: 1
DCHIP	Vertices: 9335 Edges: 8432441 Density: 0.1936 Components: 1	Vertices: 9335 Edges: 6709540 Density: 0.154 Components: 1	Vertices: 9335 Edges: 8888707 Density: 0.204 Components: 1	Vertices: 9335 Edges: 7428462 Density: 0.1705 Components: 1	Vertices: 9335 Edges: 8018211 Density: 0.184 Components: 1
DCHIP MM	Vertices: 9335 Edges: 8642740 Density: 0.1984 Components: 1	Vertices: 9335 Edges: 6928908 Density: 0.159 Components: 1	Vertices: 9335 Edges: 8447267 Density: 0.1939 Components: 1	Vertices: 9335 Edges: 8136468 Density: 0.1868 Components: 1	Vertices: 9335 Edges: 8425081 Density: 0.1934 Components: 1

Table 3.10: A p-value of 0.05 was used to threshold the graphs using all combinations of preprocessing methods and similarity metrics.

Qvalue 0.05	Pearson Correlation	Spearman Correlation	Kendall Correlation	Euclidean Correlation	Mutual Information
RMA	Vertices: 9335 Edges: 2899437 Density: 0.0666 Components: 1	Vertices: 9335 Edges: 2792321 Density: 0.0641 Components: 1	Vertices: 9335 Edges: 2852322 Density: 0.0655 Components: 1	Vertices: 9335 Edges: 2599232 Density: 0.0597 Components: 1	Vertices: 9335 Edges: 2932223 Density: 0.0673 Components: 1
MAS 5.0	Vertices: 9335 Edges: 2803158 Density: 0.0643 Components: 1	Vertices: 9335 Edges: 2874235 Density: 0.066 Components: 1	Vertices: 9335 Edges: 2917100 Density: 0.067 Components: 1	Vertices: 9335 Edges: 2508510 Density: 0.0576 Components: 1	Vertices: 9335 Edges: 2738042 Density: 0.0628 Components: 1
GCRMA	Vertices: 9335 Edges: 2964630 Density: 0.068 Components: 1	Vertices: 9335 Edges: 2663599 Density: 0.0611 Components: 1	Vertices: 9335 Edges: 3037607 Density: 0.0697 Components: 1	Vertices: 9335 Edges: 2501890 Density: 0.0574 Components: 1	Vertices: 9335 Edges: 2765716 Density: 0.0635 Components: 1
GCRMA MM	Vertices: 9335 Edges: 3013327 Density: 0.0692 Components: 1	Vertices: 9335 Edges: 2851631 Density: 0.0655 Components: 1	Vertices: 9335 Edges: 2935111 Density: 0.0674 Components: 1	Vertices: 9335 Edges: 2537408 Density: 0.0582 Components: 1	Vertices: 9335 Edges: 3004885 Density: 0.069 Components: 1
DCHIP	Vertices: 9335 Edges: 2778651 Density: 0.0638 Components: 1	Vertices: 9335 Edges: 2856427 Density: 0.0656 Components: 1	Vertices: 9335 Edges: 2923464 Density: 0.0671 Components: 1	Vertices: 9335 Edges: 2455035 Density: 0.0564 Components: 1	Vertices: 9335 Edges: 3114134 Density: 0.0715 Components: 1
DCHIP MM	Vertices: 9335 Edges: 2701979 Density: 0.062 Components: 1	Vertices: 9335 Edges: 2863578 Density: 0.0657 Components: 1	Vertices: 9335 Edges: 2724803 Density: 0.0625 Components: 1	Vertices: 9335 Edges: 2417953 Density: 0.0555 Components: 1	Vertices: 9335 Edges: 2746662 Density: 0.063 Components: 1

Table 3.11: A q-value of 0.05 was used to threshold the graphs using all combinations of preprocessing methods and similarity metrics.

Maximal Clique 2	Pearson Correlation	Spearman Correlation	Kendall Correlation	Euclidean Correlation	Mutual Information
RMA	Vertices: 2891 Edges: 60826 Density: 0.0146 Components: 137	Vertices: 6959 Edges: 56636 Density: 0.0023 Components: 254	Vertices: 8623 Edges: 697063 Density: 0.0188 Components: 149	Vertices: 7342 Edges: 479230 Density: 0.0178 Components: 186	Vertices: 2562 Edges: 26139 Density: 0.008 Components: 14
MAS 5.0	Vertices: 2784 Edges: 57281 Density: 0.0148 Components: 131	Vertices: 7325 Edges: 54514 Density: 0.002 Components: 272	Vertices: 8959 Edges: 728730 Density: 0.0182 Components: 143	Vertices: 7890 Edges: 495844 Density: 0.0159 Components: 180	Vertices: 2664 Edges: 24324 Density: 0.0069 Components: 13
GCRMA	Vertices: 3007 Edges: 64112 Density: 0.0142 Components: 133	Vertices: 6533 Edges: 53426 Density: 0.0025 Components: 243	Vertices: 8153 Edges: 656040 Density: 0.0197 Components: 152	Vertices: 7161 Edges: 466494 Density: 0.0182 Components: 190	Vertices: 2480 Edges: 25507 Density: 0.0083 Components: 15
GCRMA MM	Vertices: 2719 Edges: 56132 Density: 0.0152 Components: 127	Vertices: 7200 Edges: 54351 Density: 0.0021 Components: 269	Vertices: 8239 Edges: 716378 Density: 0.0211 Components: 141	Vertices: 7053 Edges: 498256 Density: 0.02 Components: 172	Vertices: 2418 Edges: 24776 Density: 0.0085 Components: 13
DCHIP	Vertices: 2686 Edges: 57684 Density: 0.016 Components: 133	Vertices: 7160 Edges: 55300 Density: 0.0022 Components: 234	Vertices: 8073 Edges: 651686 Density: 0.02 Components: 160	Vertices: 7561 Edges: 504800 Density: 0.0177 Components: 181	Vertices: 2437 Edges: 24560 Density: 0.0083 Components: 15
DCHIP MM	Vertices: 3038 Edges: 64058 Density: 0.0139 Components: 131	Vertices: 6799 Edges: 53217 Density: 0.0023 Components: 245	Vertices: 8137 Edges: 649108 Density: 0.0196 Components: 160	Vertices: 7106 Edges: 499065 Density: 0.0198 Components: 172	Vertices: 2669 Edges: 24097 Density: 0.0068 Components: 13

Table 3.12: The threshold value at which the number of maximal cliques doubled was used to create the graphs using all combinations of preprocessing methods and similarity metrics.

Maximal Clique 3	Pearson Correlation	Spearman Correlation	Kendall Correlation	Euclidean Correlation	Mutual Information
RMA	Vertices: 2891 Edges: 566391 Density: 0.1356 Components: 163	Vertices: 7453 Edges: 70049 Density: 0.0025 Components: 228	Vertices: 8902 Edges: 697093 Density: 0.0176 Components: 145	Vertices: 7962 Edges: 479385 Density: 0.0151 Components: 192	Vertices: 2151 Edges: 30748 Density: 0.0133 Components: 12
MAS 5.0	Vertices: 2795 Edges: 594447 Density: 0.1522 Components: 151	Vertices: 6940 Edges: 75219 Density: 0.0031 Components: 219	Vertices: 9305 Edges: 661694 Density: 0.0153 Components: 150	Vertices: 8575 Edges: 495164 Density: 0.0135 Components: 207	Vertices: 2107 Edges: 28307 Density: 0.0128 Components: 13
GCRMA	Vertices: 2802 Edges: 530087 Density: 0.1351 Components: 156	Vertices: 6998 Edges: 67412 Density: 0.0028 Components: 217	Vertices: 9409 Edges: 670895 Density: 0.0152 Components: 155	Vertices: 7468 Edges: 506725 Density: 0.0182 Components: 181	Vertices: 2050 Edges: 29544 Density: 0.0141 Components: 13
GCRMA MM	Vertices: 2706 Edges: 524467 Density: 0.1433 Components: 156	Vertices: 7775 Edges: 72665 Density: 0.0024 Components: 217	Vertices: 8516 Edges: 649989 Density: 0.0179 Components: 155	Vertices: 7529 Edges: 454152 Density: 0.016 Components: 183	Vertices: 2071 Edges: 29597 Density: 0.0138 Components: 11
DCHIP	Vertices: 2737 Edges: 535302 Density: 0.143 Components: 157	Vertices: 7112 Edges: 66103 Density: 0.0026 Components: 222	Vertices: 8674 Edges: 673802 Density: 0.0179 Components: 154	Vertices: 7607 Edges: 517209 Density: 0.0179 Components: 179	Vertices: 2105 Edges: 28404 Density: 0.0128 Components: 11
DCHIP MM	Vertices: 2669 Edges: 586377 Density: 0.1647 Components: 174	Vertices: 6864 Edges: 65578 Density: 0.0028 Components: 240	Vertices: 8214 Edges: 653873 Density: 0.0194 Components: 142	Vertices: 7369 Edges: 459961 Density: 0.0169 Components: 178	Vertices: 2025 Edges: 29522 Density: 0.0144 Components: 11

Table 3.13: The threshold value at which the number of maximal cliques tripled was used to create the graphs using all combinations of preprocessing methods and similarity metrics.

Top 1%	Pearson Correlation	Spearman Correlation	Kendall Correlation	Euclidean Correlation	Mutual Information
RMA	Vertices: 4678 Edges: 653787 Density: 0.0598 Components: 148	Vertices: 7453 Edges: 52635 Density: 0.0019 Components: 263	Vertices: 7234 Edges: 697358 Density: 0.0267 Components: 150	Vertices: 8521 Edges: 610416 Density: 0.0168 Components: 192	Vertices: 2562 Edges: 26264 Density: 0.008 Components: 8
MAS 5.0	Vertices: 4828 Edges: 613959 Density: 0.0527 Components: 139	Vertices: 6861 Edges: 49369 Density: 0.0021 Components: 269	Vertices: 7794 Edges: 661730 Density: 0.0218 Components: 140	Vertices: 7947 Edges: 578537 Density: 0.0183 Components: 187	Vertices: 2762 Edges: 25035 Density: 0.0066 Components: 9
GCRMA	Vertices: 4431 Edges: 691287 Density: 0.0704 Components: 159	Vertices: 6908 Edges: 49190 Density: 0.0021 Components: 256	Vertices: 6719 Edges: 646421 Density: 0.0286 Components: 162	Vertices: 8146 Edges: 640546 Density: 0.0193 Components: 187	Vertices: 2504 Edges: 28069 Density: 0.009 Components: 7
GCRMA MM	Vertices: 4536 Edges: 614078 Density: 0.0597 Components: 152	Vertices: 7280 Edges: 51503 Density: 0.0019 Components: 245	Vertices: 7496 Edges: 739055 Density: 0.0263 Components: 144	Vertices: 8942 Edges: 567063 Density: 0.0142 Components: 202	Vertices: 2457 Edges: 27885 Density: 0.0092 Components: 9
DCHIP	Vertices: 4312 Edges: 631208 Density: 0.0679 Components: 152	Vertices: 7047 Edges: 50316 Density: 0.002 Components: 274	Vertices: 7509 Edges: 657324 Density: 0.0233 Components: 146	Vertices: 8989 Edges: 638974 Density: 0.0158 Components: 199	Vertices: 2479 Edges: 25594 Density: 0.0083 Components: 7
DCHIP MM	Vertices: 4383 Edges: 636673 Density: 0.0663 Components: 153	Vertices: 7711 Edges: 49666 Density: 0.0017 Components: 255	Vertices: 6930 Edges: 662678 Density: 0.0276 Components: 138	Vertices: 8091 Edges: 629423 Density: 0.0192 Components: 179	Vertices: 2628 Edges: 27761 Density: 0.008 Components: 7

Table 3.14: The top 1% of all correlation values was used to create the graphs using all combinations of preprocessing methods and similarity metrics.

Top 0.1%	Pearson Correlation	Spearman Correlation	Kendall Correlation	Euclidean Correlation	Mutual Information
RMA	Vertices: 2298 Edges: 566857 Density: 0.2148 Components: 165	Vertices: 4361 Edges: 65620 Density: 0.0069 Components: 359	Vertices: 5689 Edges: 653556 Density: 0.0404 Components: 258	Vertices: 6323 Edges: 479426 Density: 0.024 Components: 268	Vertices: 1952 Edges: 26631 Density: 0.014 Components: 14
MAS 5.0	Vertices: 2209 Edges: 604714 Density: 0.248 Components: 170	Vertices: 4581 Edges: 69360 Density: 0.0066 Components: 339	Vertices: 5537 Edges: 623649 Density: 0.0407 Components: 246	Vertices: 6680 Edges: 446198 Density: 0.02 Components: 252	Vertices: 1852 Edges: 28476 Density: 0.0166 Components: 15
GCRMA	Vertices: 2206 Edges: 551210 Density: 0.2266 Components: 160	Vertices: 4653 Edges: 61966 Density: 0.0057 Components: 340	Vertices: 5848 Edges: 601767 Density: 0.0352 Components: 252	Vertices: 5824 Edges: 501309 Density: 0.0296 Components: 248	Vertices: 1884 Edges: 25825 Density: 0.0146 Components: 15
GCRMA MM	Vertices: 2359 Edges: 541415 Density: 0.1947 Components: 159	Vertices: 4255 Edges: 62404 Density: 0.0069 Components: 376	Vertices: 5505 Edges: 689073 Density: 0.0455 Components: 242	Vertices: 5863 Edges: 503864 Density: 0.0293 Components: 284	Vertices: 2006 Edges: 26017 Density: 0.0129 Components: 13
DCHIP	Vertices: 2404 Edges: 550118 Density: 0.1905 Components: 176	Vertices: 4228 Edges: 62460 Density: 0.007 Components: 337	Vertices: 5286 Edges: 677541 Density: 0.0485 Components: 242	Vertices: 6458 Edges: 508691 Density: 0.0244 Components: 262	Vertices: 1837 Edges: 27346 Density: 0.0162 Components: 13
DCHIP MM	Vertices: 2241 Edges: 529345 Density: 0.2109 Components: 156	Vertices: 4597 Edges: 70009 Density: 0.0066 Components: 333	Vertices: 5411 Edges: 672129 Density: 0.0459 Components: 278	Vertices: 6823 Edges: 491355 Density: 0.0211 Components: 286	Vertices: 2070 Edges: 27256 Density: 0.0127 Components: 13

Table 3.15: The top 0.1% of all correlation values was used to create the graphs using all combinations of preprocessing methods and similarity metrics.

Chapter 4

Clique-centric Analysis of Biological Data

4.1 Introduction

Clique-centric based algorithms are an ideal fit for analyzing high-throughput microarray data. Microarrays can be used to measure a plethora of different biological data, such as the typical -omic data (transcriptomic, genomic, proteomic). The previous chapter detailed how the microarray data was transformed from raw data into an unweighted graph. In total, 180 different graphs were created and examined using the various preprocessing, correlation, and thresholding methods. The aim of this chapter is to present graph tools used in the extraction of putative gene networks from microarray data.

To simplify the analysis, the six graphs generated in the previous chapter using Pearson correlation and Maximal Clique-2 thresholding will be examined in detail. It is important to note, however, that the following tools are applicable to any of the graphs previously generated. The tools are flexible enough to account for noise that is inherent in the data. They are also able to be tuned to be more conservative in order to reduce the number of false positives in the data.

4.2 Graph tools

Chapter 2 introduced the different decision and optimization versions of the Vertex Cover and Clique problems. These tools rely on the fact that exact solutions to the optimization versions of these problems can be generated in an efficient and timely manner. Maximum Clique Finder (*MCF*) is used to extract maximum cliques from graphs and is described in detail in [49]. *Clique Enumerator*, described in [6] is used to generate all maximal cliques from the graph. While there may be relatively few maximum cliques in a graph, there are typically millions, billions, or even trillions of maximal cliques in the same graph [38].

4.2.1 Maximum Clique

Extracting exact solutions to the the maximum clique problem is a daunting task, however, using *MCF* to exploit FPT-like methodologies makes this task feasible. The result of using *MCF* is a collection of genes, or gene products, that all co-occur under the same conditions. This putative gene network has a density of 1, that is, all genes have a high level of interaction with every other gene in the clique. The maximum clique sizes for each of the six graphs were generated and the maximum clique profile is examined in Figure 4.1. The same microarray data, using different preprocessing and correlation metrics, produced a range of maximum clique sizes from 61 to 88. Of the maximum cliques that were generated, there were 23 core genes that were members of the maximum cliques generated for each graph.

4.2.2 Maximum Clique Enumeration

The previous analysis generated the size of the maximum clique for each of the graphs and a single instance of a maximum clique. The size of the maximum clique does not provide any information on the number of maximum cliques. In fact, there are many instances where there are numerous maximum cliques, often overlapping. It

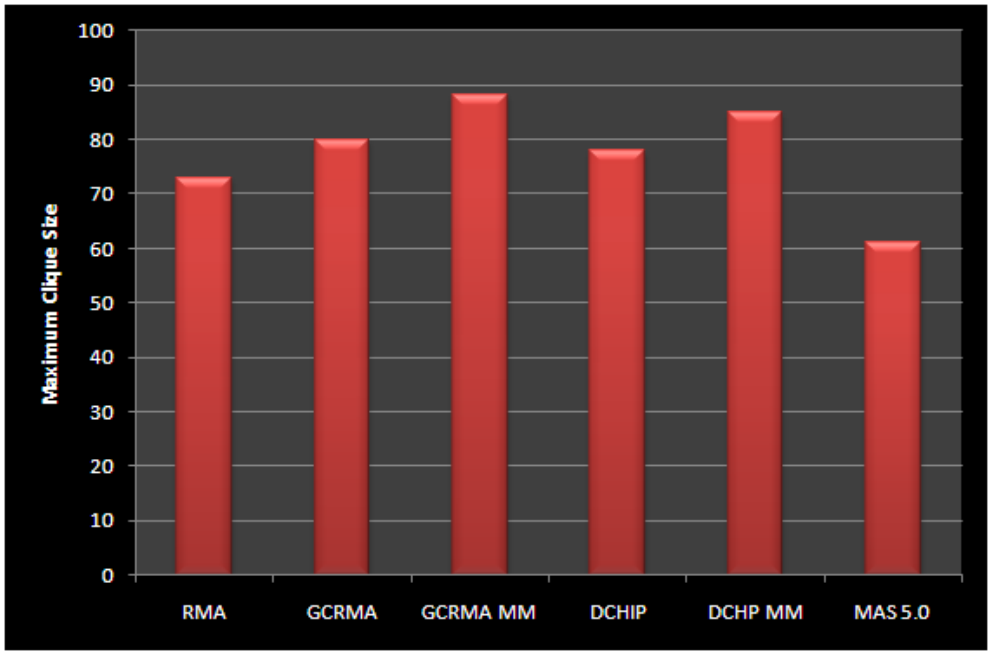


Figure 4.1: The maximum clique size for each of the graphs range from 61 to 88. The different normalization methods used to generate these graphs can have a large impact on the final results of the analysis.

has been observed that most, if not all, maximum cliques in biological data overlap, with at least one member belonging to all maximum cliques [55]. The Maximum Clique Enumeration problem asks to generate all maximum cliques in a simple, finite graph. Using the same graphs as above, all maximum cliques were generated. Figure 4.6 illustrates the the number of maximum cliques for each of the graphs.

One of the many reasons it is desirable to generate all maximum cliques in a graph is to determine if there are many disjoint maximum cliques that have different responses to experimental conditions, or, due to the pleotropic nature of genes, a few key genes are involved in a number of different responses in different putative gene networks. The percentage of the maximum clique overlap is listed in Table 4.1

4.2.3 Maximal Clique Enumeration

Recall that a maximum clique is just a special version of a maximal clique. Maximal cliques are the set of genes that work together in response to a stimuli, but unlike

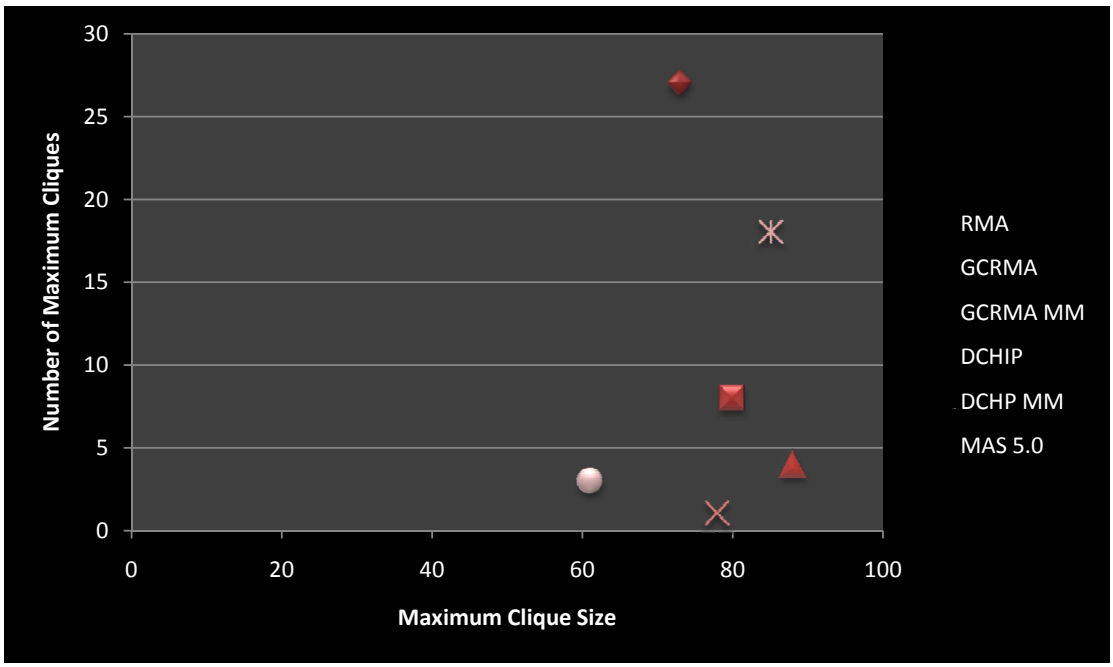


Figure 4.2: The number of maximum cliques range from 1 to 27. Note the lack of correlation between maximum clique size and the number of maximum cliques.

Normalization	Maximum Clique Size	Number of Maximum Cliques	Maximum Overlap	Minimum Overlap	Average Overlap
RMA	73	27	98.63%	90.41%	96.00%
GCRMA	80	8	98.75%	96.25%	97.86%
GCRMA MM	88	4	98.86%	96.59%	98.11%
DCHIP	78	1	N/A	N/A	N/A
DCHIP MM	85	18	98.82%	94.12%	97.29%
MAS 5.0	61	3	98.36%	96.72%	97.81%

Table 4.1: Most maximum cliques have a high level of overlap (>90%). Given the amount of overlap between maximum cliques in a single graph, it is representative of a large number of genes working together.

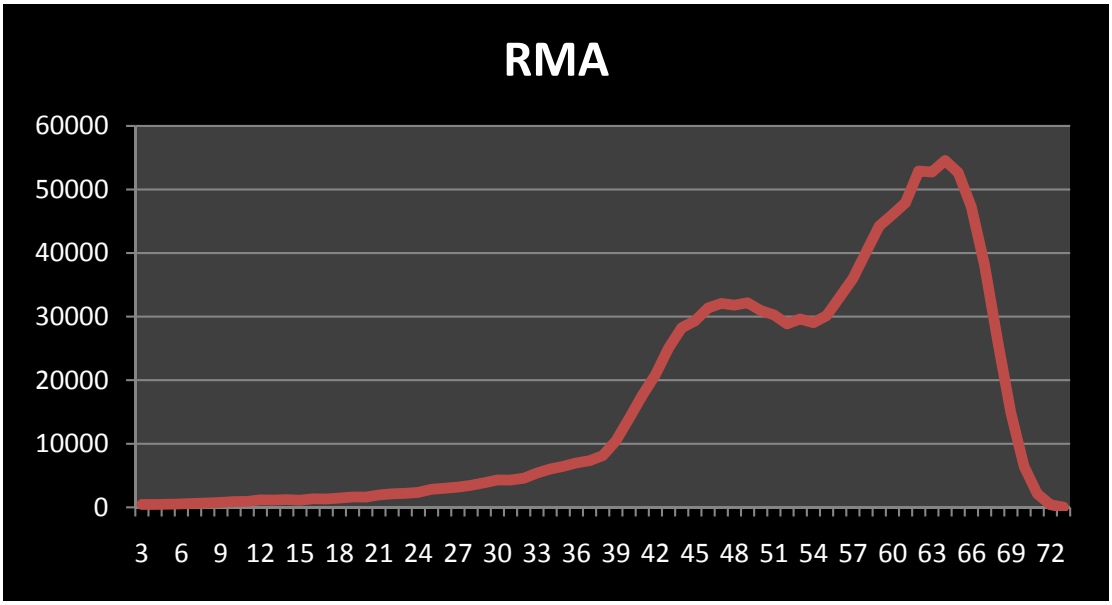
maximum clique, they are not required to be the largest response network in the graph. Maximal cliques can be thought of as a local maxima, while maximum cliques can be thought of a global maxima. Whereas the maximum cliques for the six graphs ranged in size of 61 to 88, the size of maximal cliques range from size 3 to 88. The

number of maximum cliques produced from graphs above were also relatively small, ranging from 1 to 27. The number of maximal cliques in the same graphs range from 357,444 to 1,399,801. The maximal clique profile for each of the graphs can be seen in Figure 4.3(a) to 4.3(f).

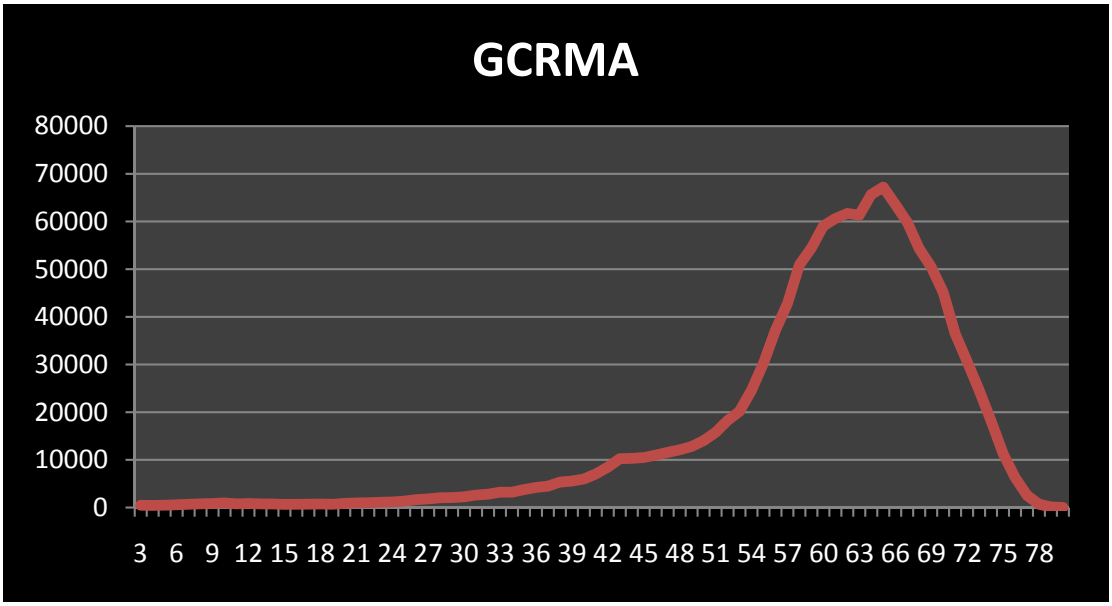
4.2.4 Paraclique

If any two cliques overlap, they can overlap at a single vertex to all but two vertices overlapping, as seen in Figure 4.4. The biological responses associated with each of these putative networks may or may not regulate the same pathways. However, in the case where two cliques are almost identical, it would be useful to be able to collapse these two networks into a single network for analysis. Missing edges in a graph could be the result of inherent noise in the data, missing data, corrupt data, or could simply be the result of the parameter selection at each step of the analysis. A single missing edge could be the difference between a larger maximum clique of size k, or multiple maximum cliques of size $k-1$, as seen in Figure 4.5. In order to correct for these anomalies, a clique-centric tool named *paraclique* was introduced in [69].

Paraclique relaxes the strict requirements of clique, which states that all possible edges must be present between all vertices, by allowing for vertices with missing edges to supplement the members of the original clique. The paraclique algorithm takes as input a simple, finite graph and generates a maximum clique. It can also be tuned to produced either overlapping or non-overlapping clusters. Overlapping clusters allow for a single gene to be in multiple clusters, while non-overlapping clusters require that a gene be in one and only one cluster. Any vertex in the original graph, but not in the maximum clique, is defined to be a candidate vertex. The list of candidate vertices is examined, one vertex at a time. The edge structure between a candidate vertex and all vertices in the maximum clique is examined. The glom factor, g, of a paraclique is defined as the allowable number of missing edges between the candidate vertex and all the vertices in the maximum clique. If a candidate vertex is missing $\leq g$ edges,

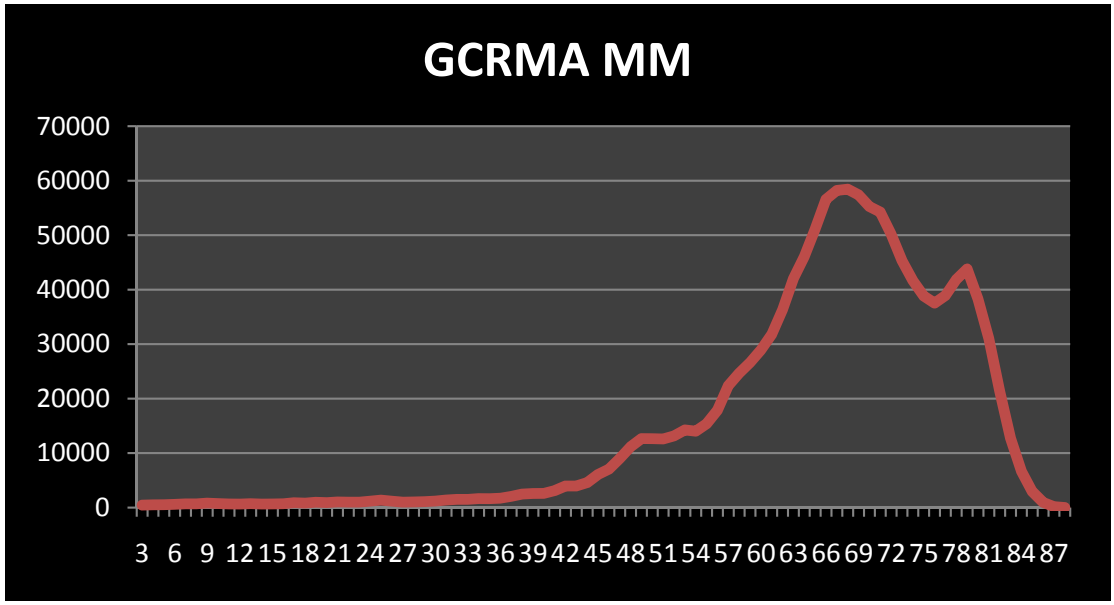


(a)

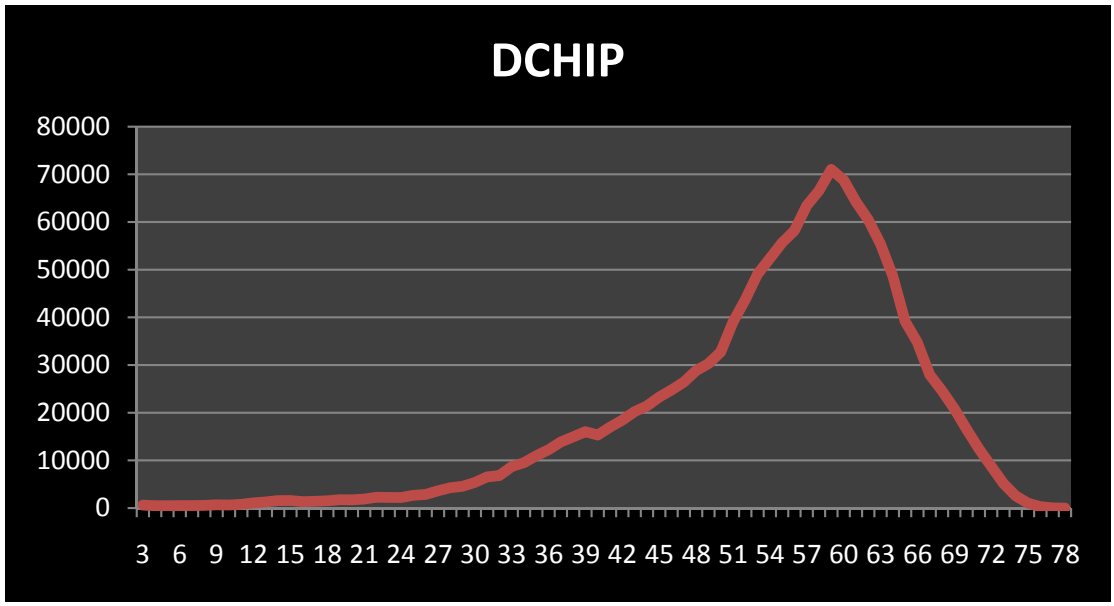


(b)

Figure 4.3: The number of maximal cliques in a graph can vary widely. The graphs that were generated using different normalization methods produced have a wide range of maximal cliques ranging from 357,444 to 1,399,801.

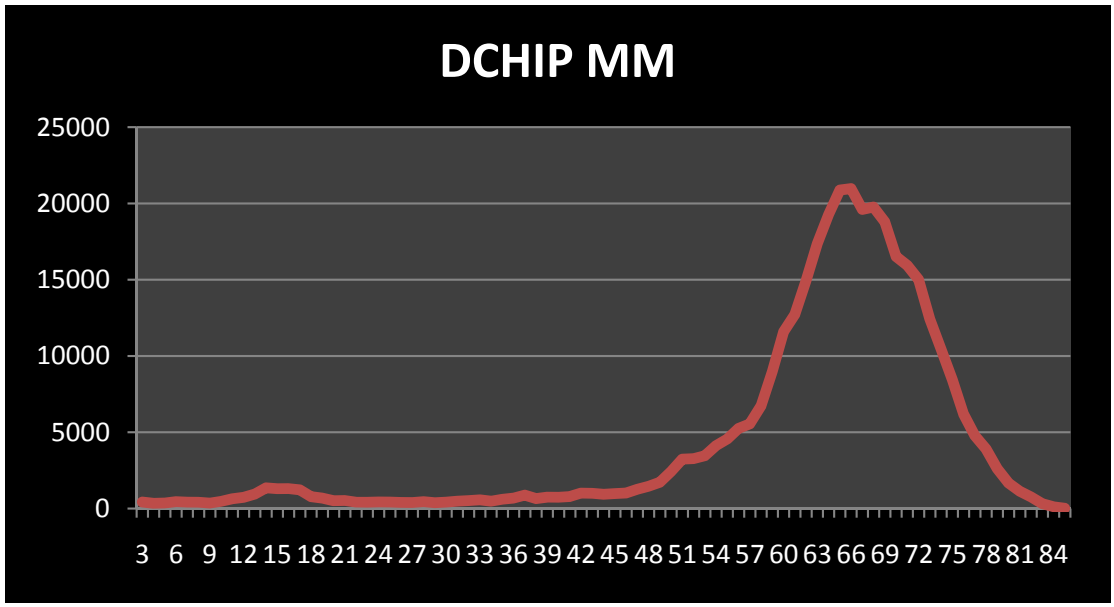


(c)

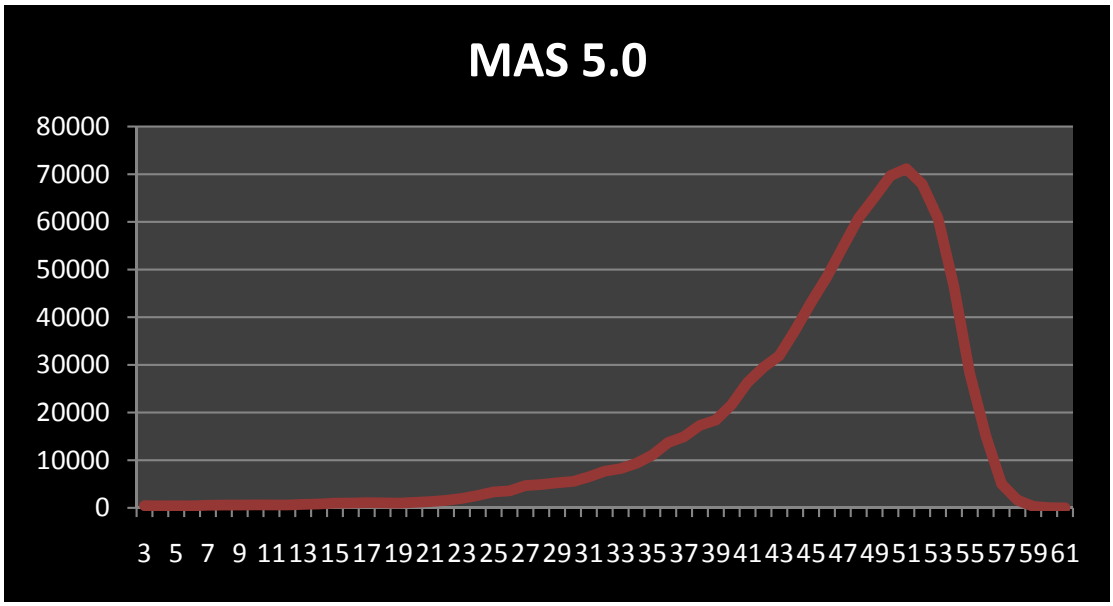


(d)

Figure 4.3: Continued.



(e)



(f)

Figure 4.3: Continued.

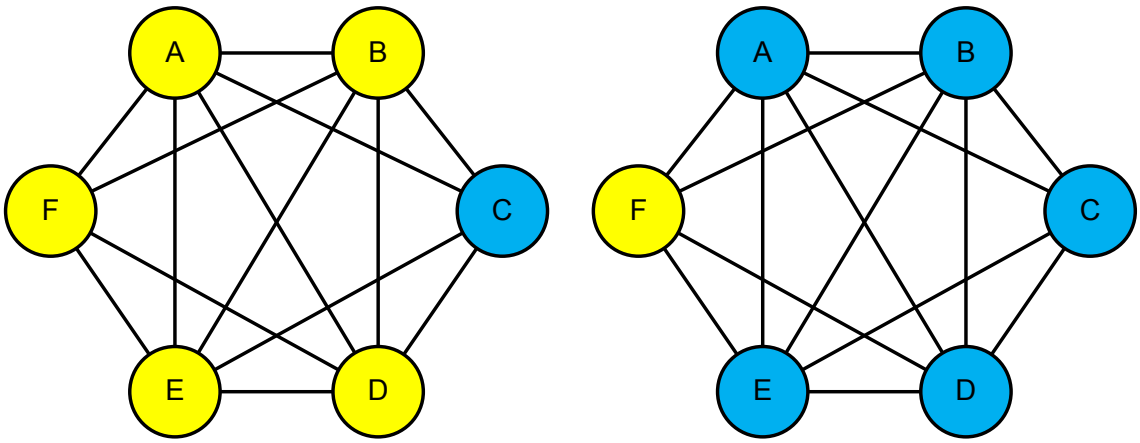


Figure 4.4: Two maximum cliques of size 5 are shown, one in yellow on the left and one in blue on the right. Note that these two maximum cliques overlap at 4 of the 5 vertices.

then it is included in paraclique result. Figure 4.6 illustrates the original clique and the resulting paraclique.

The glom factor parameter is typically set to a low value, such as 1, allowing for a minimal number of edges to be missing in our final cluster. However, in some instances it is ideal to to increase this value, or to even use a glom factor proportional to the size of the maximum clique, as proposed in [70]. When used in this manner, the number of missing edges can grow or shrink dynamically depending on the size of the initial clique. Selecting a glom factor that is too large will result in an exorbitant

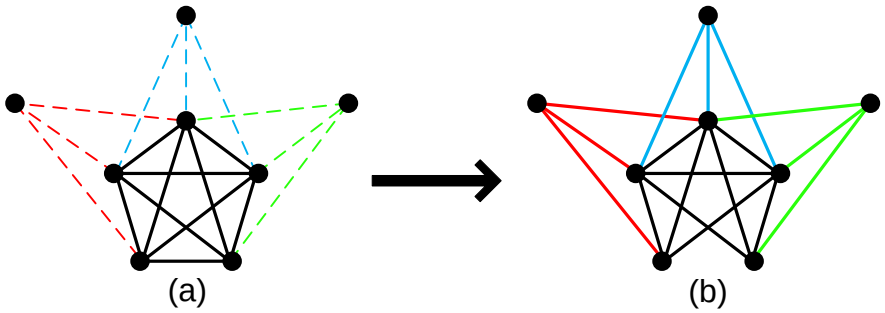


Figure 4.5: The existence, or non-existence, of a single edge can have a dramatic effect on the size and number of maximum cliques in a graph. There exists a single maximum clique of size 5 in (a). However, removal of a single edge in the graph results in 4 maximum cliques, all of size 4.

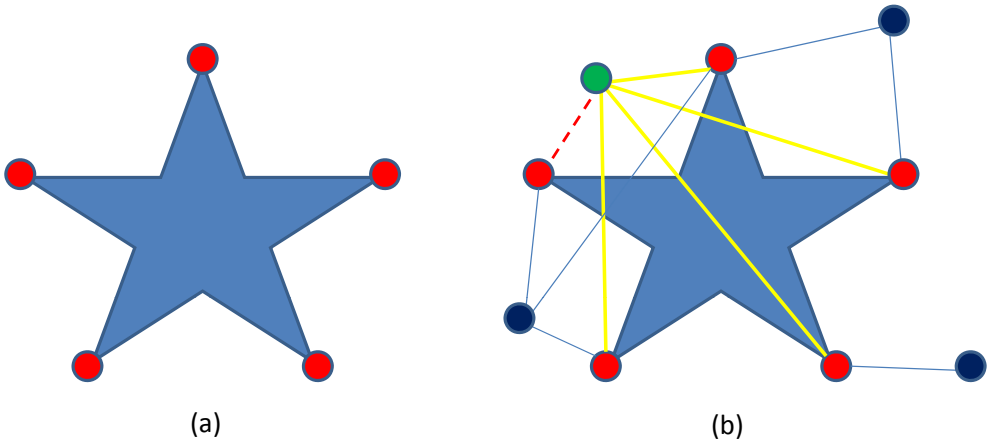


Figure 4.6: A maximum clique of size 5 is shown in part (a). After the maximum clique is generated, the remaining candidate vertices are processed to create a paraclique. The black and green vertices represent candidate vertices, with the solid yellow and blue lines representing edges between the candidate vertices and the vertices of the original clique. Using a glom factor of 1, a paraclique of size 6 is generated, with all the red vertices and the single green vertex as members (b). Only the green vertex was missing at most 1 edge (represented by the dashed red line).

amount of vertices being added to the original cluster and will introduce false positives into the cluster. Another method used to select the value of the glom factor is to determine the lowest density for the paraclique at which the results would still be considered acceptable. For instance, given that the initial clique has a density of 1, then, depending on the application, a cluster with a density of 0.90 might also be considered an acceptable. In this case, the glom factor would be derived by applying Equation 4.1,

$$g = \lceil D * (|E_c| + |V_c|) \rceil \tag{4.1}$$

where D is the required density, E_c are the edges in original clique, and V_c are the vertices in original clique. Regardless of the glom factor selection method, a glom factor must have a value of at least 1, or no edges will ever be added to the cluster to supplement the original clique results. Also, it is advisable to never select a glom

factor larger than the number of vertices in the original clique or the paraclique generated will simply be the original graph.

In addition to the glom factor, the initial clique that is given as input to the paraclique algorithm is crucial to the number and types of paracliques generated. This initial clique will from this point forward be referred to as the seed clique. One of the parameters for paraclique is the minimum size of the seed clique. A simple method of generating the seed clique is to extract the first maximum clique encountered in a graph. While this is technically a valid starting point, it is not always the best option, especially when the paraclique algorithm is tuned to produce non-overlapping clusters. Given that all maximum cliques have a density of 1, one way to rank all maximum cliques is to use the highest average correlation, HAC, for each clique. The HAC is defined to be the average correlation value for all edges in a clique. The Maximum Clique Enumeration algorithm in [55] is used to generate all maximum cliques, then a simple calculation ranks the maximum cliques by their respective HAC value. The maximum clique with the highest rank is then used as the seed clique. The largest HAC values for each of the test graphs are listed in Table 4.7

Normalization	Maximum Clique Size	HAC Values
RMA	73	0.976
GCRMA	80	0.976
GCRMA MM	88	0.978
DCHIP	78	0.977
DCHIP MM	85	0.982
MAS 5.0	61	0.969

Figure 4.7: The maximum cliques are ranked in order by their HAC value, and the maximum clique with the largest HAC value is selected to be the seed clique for the paraclique algorithm. The HAC value is just one way to rank the importance of the seed clique with respect to one another.

Although this is an improvement on the method of assigning the first maximum clique as the seed clique, it does not guarantee that the paraclique produced has the highest density. Therefore, a more detailed analysis is needed that will produce the best paracliques at each step with respect to density. First, all maximum cliques must be generated. Then, in parallel, all maximum cliques are used as seed cliques, and the first round of paracliques are generated. Before advancing to the second iteration of the paraclique algorithm, the density of the resulting paracliques are examined, and only the paraclique with the highest density is retained. At this point all other paraclique algorithms are stopped, and the remaining algorithm proceeds to the next iteration. The process of generating all maximum cliques and running parallel instances of the paraclique algorithm with these new maximum cliques as the seed cliques is repeated until the paraclique algorithm completes. The algorithm stops when no more paracliques can be generated with a size greater than a user-defined *minimum paraclique size* parameter. Note that this analysis isn't restricted to using only the HAC value and density to rank the seed cliques and the paracliques, but any combination of graph metrics could be used.

Paraclique results are generated for the graphs above using a glom factor of 1, the maximum clique with the largest HAC value as the seed clique, keeping the best paraclique, in terms of density at each step, and tuned to produce non-overlapping paracliques. The paraclique algorithm is stopped after no seed cliques of size 10 exists or no paracliques of size 10 or greater can be generated. The results are in Figure 4.9. With respect to the largest paraclique produced for each graph, the densities of these paracliques ranged from 0.99 to 1, resulting from the addition of 0 to 4 vertices. Biological graphs have been observed to usually contain a single large connected component [55], and many smaller disconnected components. It is common to find a few large non-overlapping paracliques and a vast majority of the non-overlapping paracliques to be of relatively smaller size.

4.3 Anchored Analysis

All of the aforementioned clique-centric algorithms can be tuned to include prior knowledge of gene interactions. These algorithms take as input a simple, finite graph along with a set of genes of interest. Using the genes in this set, the induced neighborhoods of these genes of interest are extracted. This new graph is said to have *anchors*, the genes of interest, and to be *anchored* at these genes. This type of anchored analysis is useful when genes with known interactions to a stimuli are not included in maximum cliques in the graph and a more in-depth analysis is needed. Due to the nature of the anchored analysis and the type of microarray data used in the current test graphs, results of applying an anchored analysis to a different biological dataset is presented later in this chapter.

Normalization	Paraclique Density
RMA	0.9979
GCRMA	0.9991
GCRMA MM	0.9992
DCHIP	1
DCHIP MM	0.9989
MAS 5.0	0.9994

Figure 4.8: Paraclique can use the results of a single iteration to improve the results for the rest of the analysis. For instance, the RMA graph has 27 different maximum cliques. 27 parallel instances of paraclique are initiated, each using a different maximum clique as a seed clique, and the resulting paraclique with the highest density is extracted and this instance of paraclique continues while the remaining 26 instances are killed. The densities of the selected paracliques for each graph are reported.

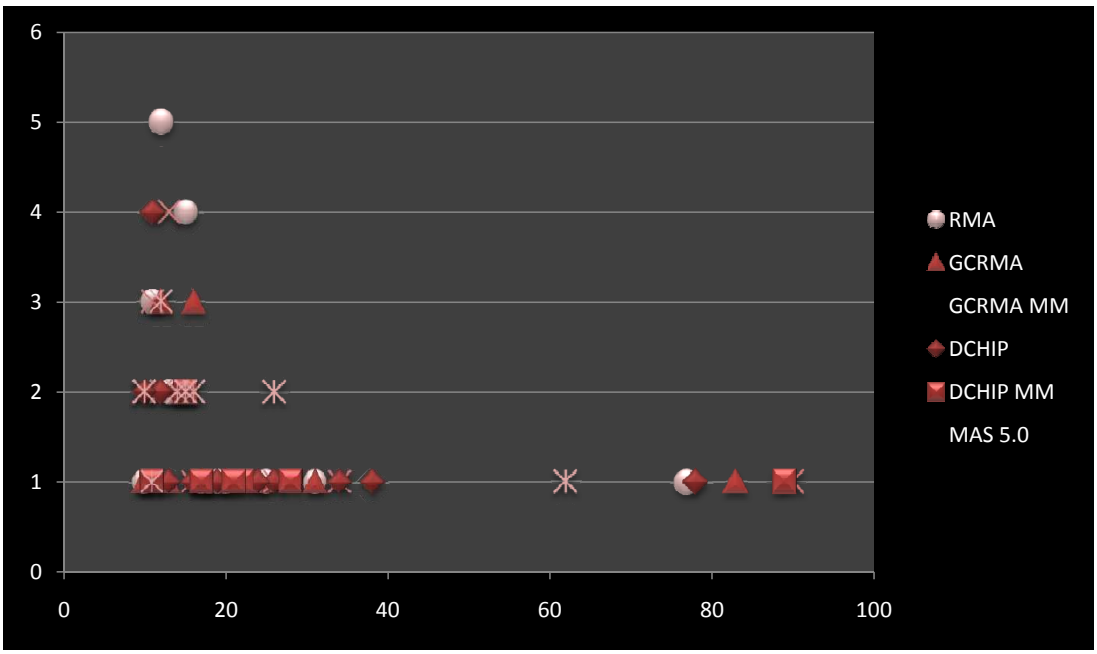


Figure 4.9: Paraclique typically generates few large, dense clusters, and many smaller, less dense, clusters. The largest paracliques for each graph are illustrated on the right hand side of the graph. Note that there are numerous smaller paracliques with size 15 to 35.

4.3.1 Anchored Maximum Clique

An anchored maximum clique is defined to be the largest clique in the graph which *all* anchors are members. Depending on the anchors in question, the graph being analyzed may require its parameters be adjusted to include all of the anchors, namely the threshold must be lowered. For example, the thresholds selected for the test graphs ranged from 0.94 to 0.96. It is feasible that one of the anchors would not have a single correlation value above 0.94, or it is not correlated to all of the other anchors at the selected threshold. To adjust for this case, the threshold levels would need to be lowered to the threshold level where all anchors are correlated with one another. Once the graph contains all edges between all anchors, the maximum clique analysis is initiated. Note that any resulting maximum clique from this graph is guaranteed to return a maximum clique with all anchors as members.

4.3.2 Anchored Maximum Clique Enumeration

The anchored maximum clique enumeration algorithm takes as input the same graph generated for the anchored maximum clique algorithm. However, instead of simply computing the size of the largest clique in the graph that contains all of the anchors, the entire set of maximum cliques that contain all of the anchors are enumerated. Using a similar approach as the regular maximum clique enumeration, the tools in [55] are given as input the newly created graph and only maximum cliques that contain all of the anchors are retained for further analysis.

4.3.3 Anchored Maximal Clique

An anchored maximal clique is defined to be the largest clique that contains all anchors and is not a subgraph of a larger clique. Like the anchored maximum clique, the threshold of the graph may need to be readjusted in order to included all edges between the anchors. Once an appropriate graph is in place, the anchored maximal clique analysis uses a Bron-Kerbosh [54] based algorithm to generate all maximal cliques using the anchors as the starting point for the algorithm. Using the anchors as the starting set guarantees that any maximal cliques generated included the anchors. The algorithm then examines all possible candidate genes to determine if any candidate gene be added to increase the current clique. Once the algorithm reaches the point that no more candidate genes can be added to the current clique, a maximal clique is produced. The last gene to be added to the maximal clique is removed and the process is repeated until all possibilities have been exhausted.

4.3.4 Anchored Paraclique

An anchored paraclique is defined to be a dense cluster that contains all of the anchors and have at most g missing edges between any vertex and any other vertex in the cluster. Given that the set of anchors have a known interaction, an extra stipulation is put into place that requires all anchors to have edges to one another, whereas

in a typical paraclique any edge is allowed to be missing. Similar to the standard paraclique analysis, an anchored paraclique will have a seed clique as input. This can be generated using the anchored maximum clique algorithm or using a more advanced method involving the anchored maximum clique enumeration algorithm to generate all anchored maximum cliques and selecting the seed clique that produced the best result, similar to the regular paraclique analysis in the previous chapter.

4.4 Results

Up to this point all of the graph-based analyses have been focused on data generated from a Yeast dataset. The following analysis is completed on a dataset derived from *Mus musculus*, a more complex organism that has approximately 25,000 genes which is a 4-fold increase over Yeast. *Mus musculus* is an important organism to researchers because they are very similar to *Homo sapiens*. The dataset is publicly available for download from the GEO website under the series GSE19935 [4]. The dataset consists both immunophenotype and gene expression data. This dataset was analyzed in the same manner as the Yeast dataset, however, this dataset was also analyzed using methods based on the biclique algorithm presented above.

4.4.1 Data Generation*

A total of 41 BXD strains were immunophenotyped, measuring the proportion of a variety of cells in the blood including the circulating T cells (%CD3), $CD4^+$ T cells (%CD4), $CD8^+$ T cells (%CD8), and B cells (%CD79). The log ratio of T cells to B cells (LN T:B), along with the log ratio of CD4 to CD8 (LN CD4:CD8) cells were computed. The gene expression data for 38 BXD strains, 34 which were also immunophenotyped, were measured using the Illumina WG-6 v1.1 Beadchip array. The gene expression data was normalized using the lumi [71] package in R with the

*This analysis was previously published in [4]

variance stabilization and robust spline normalization parameters used. The data was then filtered to retain only the transcripts that had at least half of the samples with a detection p-value of 0.25, resulting in a dataset of $\tilde{2}$0,000 transcripts. These transcripts were then correlated using Pearson correlation. The histogram of these coefficients are depicted in Figure 4.10. The resulting graph was thresholded using a q-value of 0.05. Next, all pairwise Pearson correlation coefficients were computed between the gene expression data and the immunophenotype data. A bipartite graph was then generated using the vertices from the gene expression data as one set, and the immunophenotype measurements as the second set. Edges in this graph were thresholded with a p-value of 0.001.

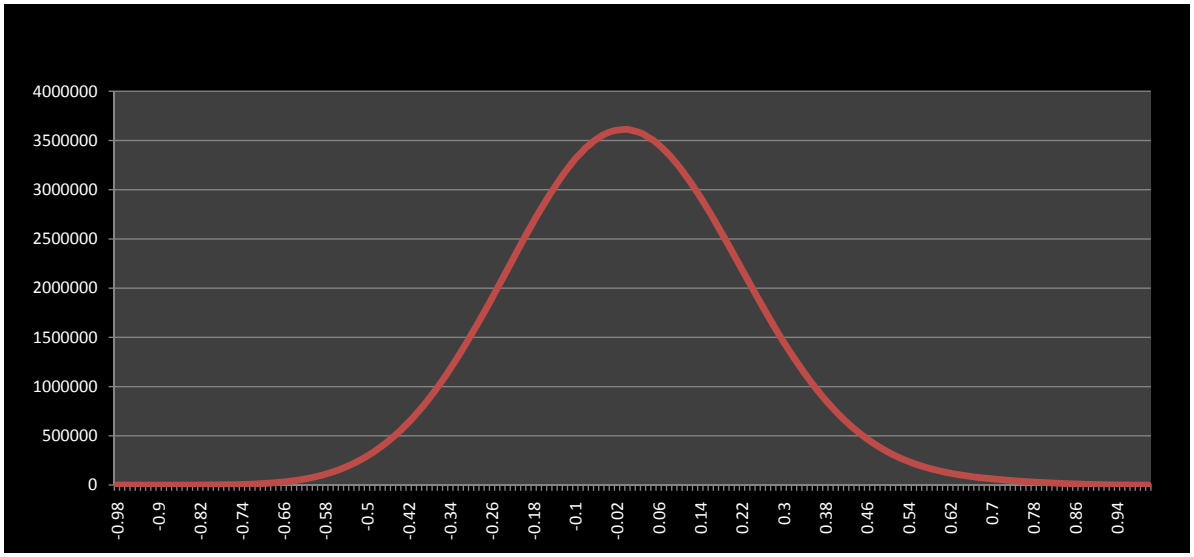


Figure 4.10: The Pearson correlation coefficients were computed pairwise among the 20,000 transcripts. The data is normally distributed with a slight positive bias, as is typical with this type of data.

4.4.2 Anchored Maximum Clique

The anchored maximum clique algorithm was used to extract maximum cliques anchored at *Acp1* and *Ptprk*. These genes were previously identified as quantitative trait transcripts in [4] The anchored maximum clique analysis used both genes

independently of each other, resulting in two independently anchored graphs, one around *Acp1* and another around *Ptprk*. The histogram of the Pearson correlation coefficients are presented in Figures 4.11 and 4.12, respectively. The anchored maximum clique algorithm extracted a maximum clique of size 500 when anchored at *Acp1*. The correlation coefficients are in the range $|r| = [0.515,0.917]$. The genes in this maximum clique were analyzed using GO enrichment and were found to be involved with cell cycle, cell division, and DNA replication. The anchored maximum clique algorithm generated a maximum clique of size 297 when anchored at *Ptprk*. The correlations in this maximum clique fell in the range $|r| = [0.474,0.68]$. These genes were also analyzed using GO enrichment, however no significant GO enrichment was observed.

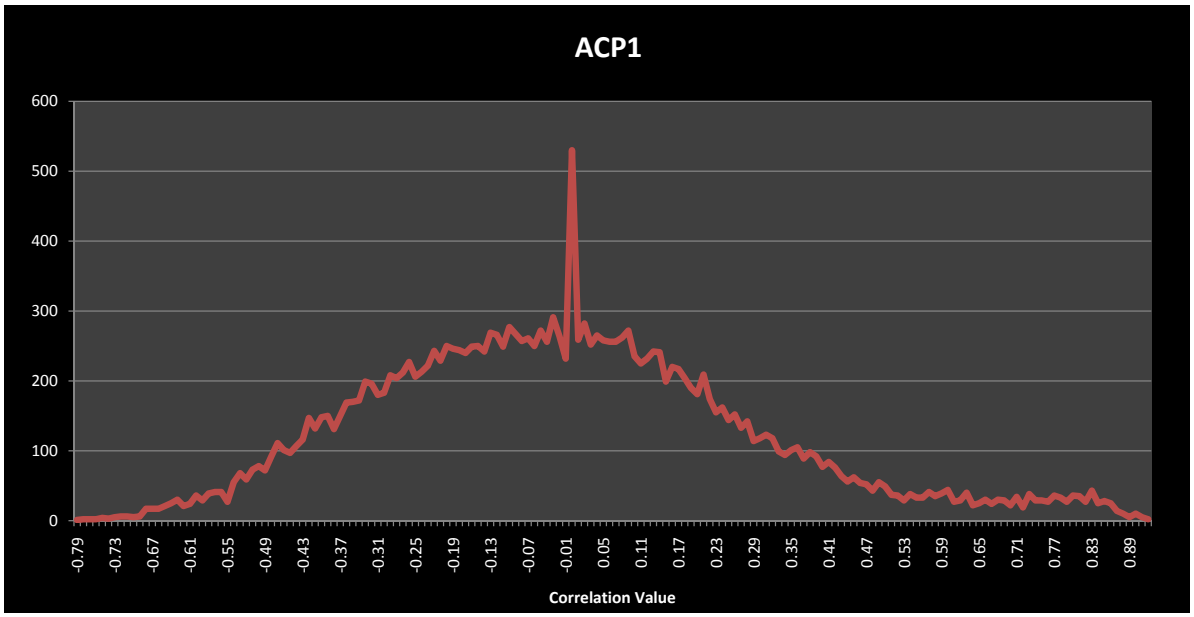


Figure 4.11: The Pearson correlation coefficients were computed pairwise among the 20,000 transcripts. The data is normally distributed with a slight positive bias, as is typical with this type of data.

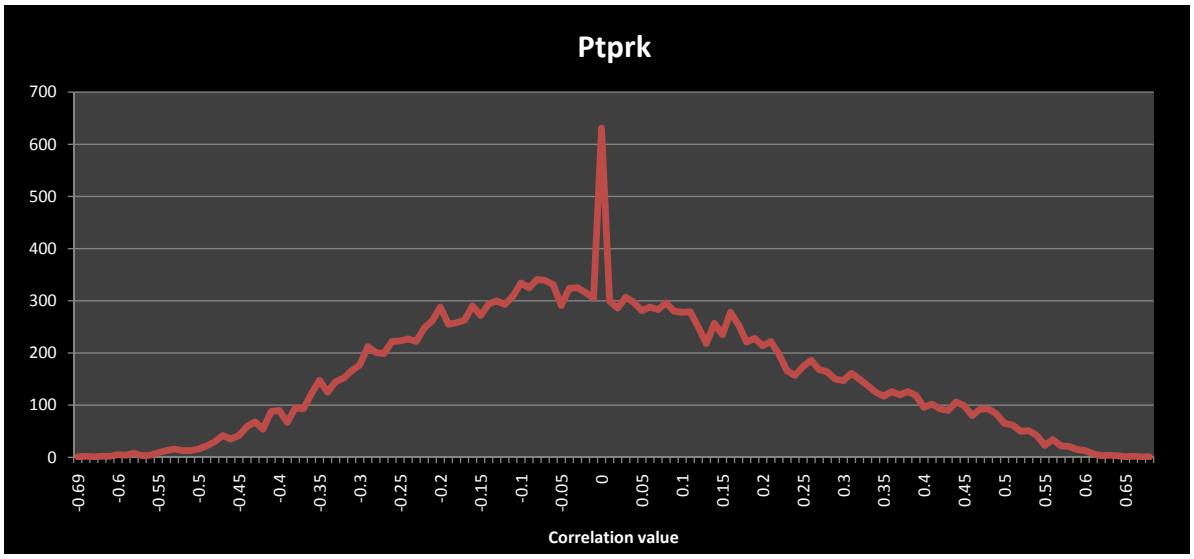


Figure 4.12: The Pearson correlation coefficients were computed pairwise among the 20,000 transcripts. The data is normally distributed with a slight positive bias, as is typical with this type of data.

4.4.3 Biclique

The bipartite graph, described above, was created to identify the relationship between gene coexpression networks and immune function. Five of the phenotypes were retained in the biclique analysis including %CD4, %CD8, %CD3, LN T:B, and LN CD4:CD8. All of the gene expression values described above were also used in the biclique analysis. The resulting bipartite graph was analyzed using the biclique algorithm presented previously to generate both the size of the maximum clique and all maximal cliques. Figure 4.13 shows all maximal bicliques generated. The maximal clique that interacts with the largest number of immunophenotypes includes 4 immunophenotypes and 14 transcripts. The maximal biclique with the most edges interacts with only 2 immunophenotypes and 80 transcripts.

4.5 Conclusion

Two analyses of biological data by clique-centric tools were presented. The Yeast dataset was converted into graphs using the RMA normalization method and

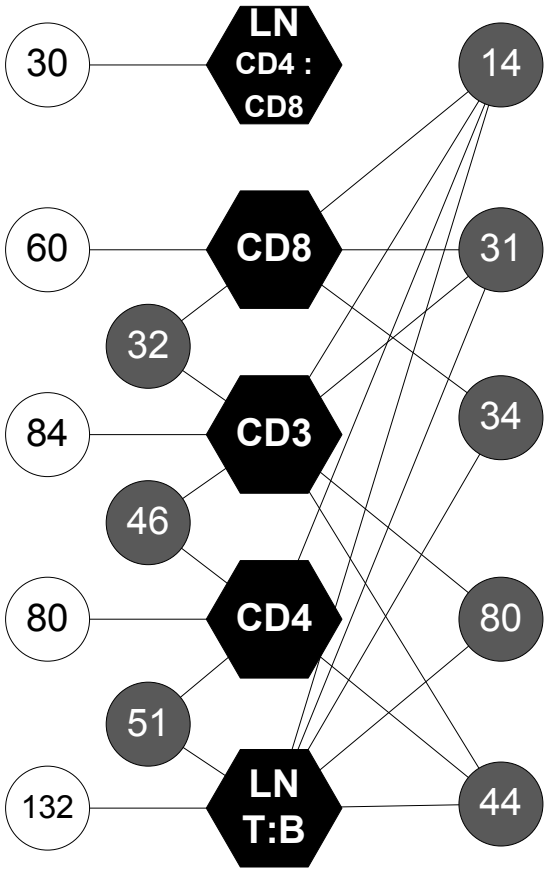


Figure 4.13: The resulting maximal bicliques are represented using the 5 immunophenotypes and 20,000 transcript expression. Immunophenotypess are listed in the center of the graph and are symbolized by hexagons. The number of transcripts in the maximal bicliques are listed in the circles. The white circles represent maximal bicliques with a single immunophenotype. The gray circles represent the transcripts that belong to a maximal biclique with more than 1 immunophenotype[4].

processed using Pearson correlation. The resulting graph contained maximum cliques with sizes ranging from 61 to 88. The number of maximum cliques in these graphs ranged from 1 to 27. Another clique-centric tool, paraclique, was used to analyze the different Yeast graphs. This tool corrects for noise inherent in the data and softens the strict requirements of the clique algorithm. The resulting paracliques ranged in size from 62 to 90, using a glom factor of 1. The effects of the graph creation parameters, such as the normalization method and similarity metric, are seen by the size and

number of maximum cliques in the graphs. This also extends to other clique-like structures, such as paraclique.

An analysis that included anchor genes and a bipartite graph was also presented. This analysis used a more complex dataset generated from *Mus musculus*. The approach of using anchors in the analysis allows for previous knowledge to supplement the clique-centric algorithms. The biclique analysis allows for data of different types to be integrated together and analyzed meaningful network interactions. A network involving gene expression and immunophenotypes was identified with positive results.

Overall, the use of clique-centric algorithms to analyze biological datasets has been shown to be feasible and to produce extremely dense clusters. Most of the clique-centric tools presented above are guaranteed to generate clusters with a density of 1, with a few generating clusters with a density near 1. It is important to note that very few clustering algorithms can consistently produce clusters with densities in this range [49].

Chapter 5

Out-of-Core Methods

A preliminary version of this chapter was first published in *2009 IEEE/ACS International Conference on Computer Systems and Applications*:

G. L. Rogers, C. A. Phillips, J. D. Eblen, A. D. Perkins, F. N. Abu-Khzam and M. A. Langston. Using Out-of-Core Techniques to Produce Exact Solutions to the Maximum Clique Problem on Extremely Large Graphs, in 2009 ACS/IEEE International Conference on Computer Systems and Applications, 2009.

Only minor modifications have been made to the published work. My contribution to this paper was implementing the out-of-core algorithms, running all of the experiments, and the majority of the writing and revisions.

5.1 Introduction

New microarray technologies generate an abundant amount of data, which must be analyzed in a timely manner. The first microarray experiment published in *Science* measured only 45 probes, while current microarray chips measure on the order of 50K probes for genes and millions of probes for SNPs. Given the state of the technology today, and the prediction of the new technologies on the horizon, changes must be implemented in the current methodology of analyzing high-throughput microarray data. One of the changes that must be made is the creation of algorithms that can

handle extremely large datasets that are too large to store in core memory. With high-performance computing moving towards a more distributed memory working environment, algorithms that once ran on large monolithic memory machines must be rewritten to exploit the new memory hierarchy. This chapter reviews algorithms that builds upon the current analysis of biological data using clique-centric tools and will examine some of the pitfalls of analyzing large datasets.

5.2 Motivation

As newer technologies produce cleaner and more accurate data, the size of data produced increases. The type and amount of biological data is growing at such a rate that the methods to analyze and store the data is becoming more important everyday. Current microarray technologies measure around 50K probes for gene expression data, and approximately 2 million probes for SNP data. These datasets measure anywhere from a few hundred megabytes (MB) to a few gigabytes (GB) in size. However, new technologies, such as Next Generation Sequencing, produce datasets that are measured in terabytes (TB).

The concern with analyzing this amount of data doesn't only effect the time it takes to process the data, but the lack of ability for current algorithms to handle this much data. Even if the algorithms were designed to handle these datasets, the average computer system isn't equipped with the proper hardware for this type of analysis. With respect to the aforementioned graph tools, the amount of memory required to store and analyze these datasets grows polynomially, since each gene-gene pair must be correlated. Figure 5.1 illustrates the amount of correlations required for varying number of probes on microarrays.

It is easy to see that with this type of growth in the size of raw data being produced, that a new approach is necessary to analyze the vast amount of data. The hardware requirements are examined in section 5.3. The out-of-core algorithm

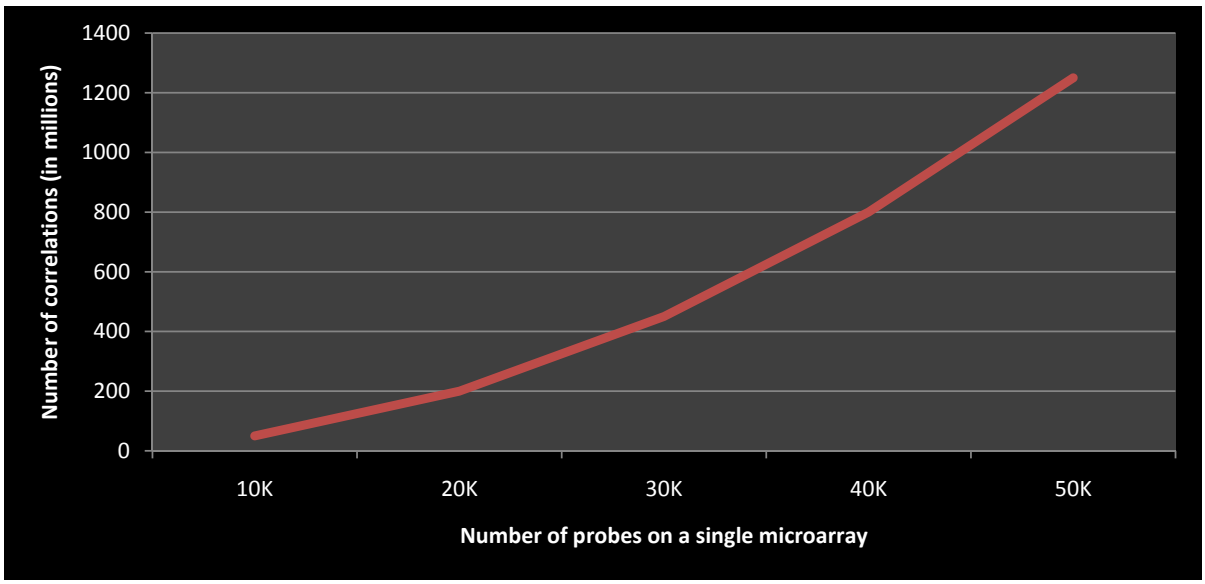


Figure 5.1: The number of correlations, or edges in a graph, grows polynomially with respect to the size of the input graph. While current computational power is sufficient for analyzing graph with approximately 50K probes in a timely fashion, the advent of newer technologies will produce datasets that so large in size that they will need special hardware and algorithms in order to be analyzed.

for computing exact solutions to the maximum clique problem is then introduced. Finally, the results of running these out-of-core algorithms on real data are analyzed.

5.3 Hardware Considerations

As computer hardware changes, so must the algorithms that run on them. Datasets have increased in size over the years, and one solution to analyzing these larger datasets was to get access to a larger machine that had more processing power and more memory, such as the Altix 3700. However, in recent times high-performance computing has shifted from large monolithic memory machines to a more distributed memory model. Algorithms that once had access to several TB of shared memory, now only have access to a few GB of local memory per core. This prohibits the size of the datasets that algorithms can analyze at a single time point. Compute clusters also have grown in popularity in labs across the world, and these clusters share the same

type of distributed memory design as the high-performance computers. Therefore, it is necessary to redesign algorithms to take into account the new hardware requirements in this new computing environment.

Memory hierarchy has a large impact on the running time of any algorithm. The faster data can be transferred to the CPU registers to be processed, the faster the analysis will be completed. The typical memory hierarchy is depicted in Figure 5.2. It is generally accepted that as you increase the levels of memory from disk to registers, that the memory gets more expensive and there is less of it. Therefore it is a necessity to design algorithms that would take advantage of this concept. However, cases do arise where the data is too large to fit into core memory, therefore out-of-core memory algorithms must be implemented to analyze the data.

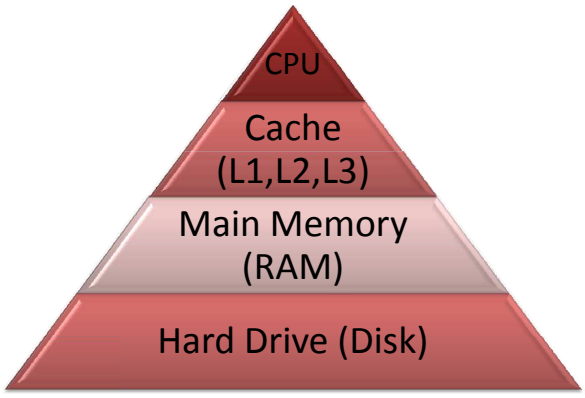


Figure 5.2: Memory speed is typically inverse with the amount of memory available. The larger, slower disk is an inexpensive means of storing data long term, however, sometimes it is necessary to use this slow memory in order to analyze large datasets.

In addition to the size of memory on machines, other issues to consider would be the failure of hardware such as disk or network controllers. The question of hardware failure isn't *if* the hardware will fail, but *when* it will fail. Mean time to failure, MTTF, is the industry standard as the expected lifespan of a given piece of hardware. Consumer grade hard drives currently ship with a MTTF between 1,000,000 to 1,500,000 hours, which translates into a failure rate roughly less than 1%. However, [72] shows that at several computing locations, these failure rates were typically

between 2 and 4%, and sometimes were as high as 13%. Given that compute nodes are simply collections of smaller hardware components, it is a statistical certainty that it will fail sometime, and software should be designed to handle such failures. These and other hardware issues are considered when implementing the out-of-core algorithms

5.4 Sampling the Data to Fit in Core Memory

One method to try to analyze the data using existing hardware and software is to sample the data so it fits in core memory. To test the hypothesis that taking random samples of the data does not drastically effect the results of the graph analysis, sample sizes of 3,000, 6,000, and 9,000 genes were taken from the Yeast dataset and compared against the original dataset of 9,335 genes. All of the graphs were generated by using the RMA normalization method, Pearson correlation, and the Maximum clique-2 thresholding algorithm. Recall that the original dataset contained a maximum clique size of 73, and there were 27 different maximum cliques of this size. The resulting maximum clique sizes, along with the number of maximum cliques of that size, are listed in Table 5.1. One can conclude that taking samples of the data, even at a very high sampling rate, has an affect on the structures found within the graph. Therefore, it is unadvisable to use sampling in the analysis of biological graphs.

5.5 Available Software to Analyze Large Graphs

Software packages Pregel[73] and Pegasus[74] were reviewed to determine if they met the basic requirements for the analysis presented in this dissertation. Pregel was developed by Google to be a scalable software platform on which large-scale graphs can be analyzed. The framework is similar to that found in the Map/Reduce model, but with a few improvements that help reduce both I/O and internode communication. While Pregel looks promising for the analysis of large-scale graphs,

Sample Size	Maximum Clique Size	Maximum Clique Count
3000	48	2
6000	51	92
9000	55	8
9335	73	27

Table 5.1: The maximum clique sizes derived from sampling the data at intervals 3000, 6000, and 9000. Note that both the size and the count of the maximum cliques are affected by the sampling.

this framework is not well suited for the Maximum clique problem. It is, however, well suited for analyzing graphs properties such as PageRank, graph connectedness, and shortest path problems. Pegasus was developed at Carnegie Melon University to analyze large graphs. This software is build on the open source Map/Reduce software, Hadoop. Much like Pregel, this software allows the user to analyze large graphs in an efficient manner, however, the software is limited to a handful of algorithms, none which are near the complexity of the maximum clique problem. Pegasus provides methods for extracting the following graph properties: degree structure, PageRank, random walks with restart, radius, and connected components. A few other software packages were reviewed, such as GoldenOrb, but they too lacked the necessary tools. Therefore, it is necessary to implement new algorithms that will generate results for the graph based analysis presented in this dissertation on large graphs.

5.6 Out-of-Core Maximum Clique Algorithm

Novel algorithms are presented that generate *exact* solutions to the maximum clique problem for graphs that are too large to fit within core memory. A combination of in-core and out-of-core techniques are exploited to dissect these large graphs into smaller

and more manageable segments. A global solution to the maximum clique problem is extracted from the set of local solutions generated for each of the smaller segments. Parallelizing the search for the maximum clique size within these components is essential to improving the overall run times for these algorithms.

Approximations to the maximum clique problem using out-of-core approaches have previously been studied. See, for example,[75, 76]. However, given the expense of generating data, approximations to the maximum clique problem can be poor substitutes and thus this dissertation focuses entirely on finding exact solutions [77, 78].

The custom software package, MCF [49], is used as the foundation for the proposed algorithms. MCF takes a simple, finite graph as input and returns the maximum clique size. MCF uses bit adjacency matrices to store graphs efficiently in core memory, while keeping the ability to check a multitude of data items, including the existence of edges between pairs of vertices and common neighbors for two or more vertices. Figure 5.3 illustrates the advantage of storing graphs as bit matrices as compared to integer matrices. The MCF is based on algorithms for vertex cover (VC) derived from previous work as reported in [77]. Preprocessing and branching serve as the basis for the computational approach used in MCF. To utilize the MCF software, the graphs are dissected into segments small enough to store in main memory. Depending on the size of the graph, one of the following cases will apply:

- Case 1. The graph can be stored in main memory as a bit adjacency matrix. This case is straightforward and can be solved using current tools and existing hardware.

- Case 2. The graph cannot be stored in main memory as a bit adjacency matrix, but is sparse enough to be stored as an adjacency list. In this case we use the edges-in-core (EIC) algorithm, to follow.

- Case 3. The graph is too large to store in main memory even as an adjacency list, but we can store its vertex list in main memory. Thus the edge list must be

stored in external memory, where access times are orders of magnitude slower than main memory. In this case we use the edges-out-of-core (EOC) algorithm, to follow.

- Case 4. The graph is so large that even its list of vertices will not fit into main memory. This case is beyond the scope of this dissertation and unlikely to be encountered in practice.

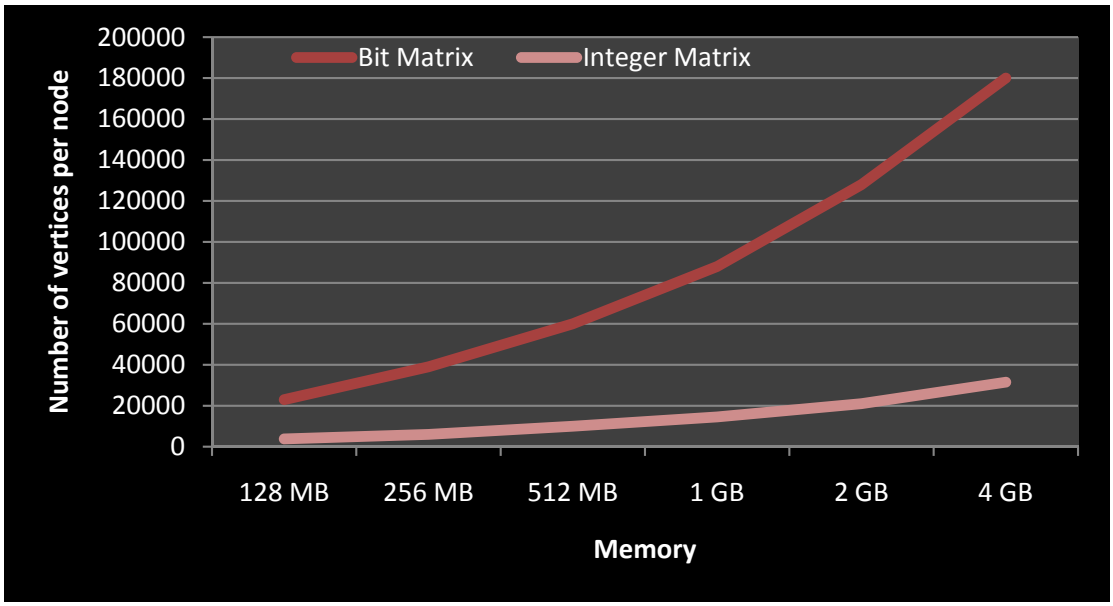


Figure 5.3: Storing the graph as a bit matrix not only allows a larger graph to be stored in core memory at a single time point, but also allows for fast bit operators on the graph.

The EIC and EOC algorithms take extremely large simple, finite graphs as input and return the maximum clique size as output. Both algorithms break the graph into a multitude of smaller segments and solve maximum clique for each segment. The algorithms use a distributed memory model to parallelize the runs of the MCF software on each of the smaller segments, independently. Distributed memory systems are capable of message passing, such as MPI [79, 80], between nodes that have their own private memory space. The proposed model dictates that messages are sent between a single node, that is defined as the master node, and each of the other

nodes, that are defined as worker nodes. MCF currently returns only the size of the maximum clique found, but can easily be extended to return the elements of the maximum clique, if so desired.

The master node is responsible for preprocessing the graph, identifying connected components, and constructing workloads to send to the worker nodes. Given N worker nodes in the system, each with M bytes of memory, there will be N bins, each capable of containing a subgraph of the original graph that is less than M bytes in size. Each bin may contain a mixture of connected components and any number of vertices, along with their induced neighborhoods, as long as the memory constraints are not violated. Bin construction techniques differ between the two proposed algorithms, EIC and EOC, and will be described below.

After the primary invocation of a worker node, an initial request for work is sent to the master node. Once the master node receives the request for work, a workload is generated by the master node and sent to the waiting worker node. The worker node then uses a local instance of MCF to compute the size of the maximum clique for its workload. The worker node then returns its local solution to the master node, and then requests more work. The master node processes the entire graph until the search space is exhausted. Once the entire graph is processed, the master node sends a broadcast message to all worker nodes to finalize and exit.

Reducing the size of the graph via preprocessing is an essential first step in solving maximum clique via the proposed algorithms. Both the EIC and the EOC algorithms preprocess the graph to eliminate vertices that cannot be members of a clique larger than the current maximum clique size (CMCS). One way to eliminate these vertices is to remove all vertices that have a degree less than CMCS-1. By default, the CMCS is initially set at 2, representative of an edge between two vertices, and increases when a new maximum clique size is returned to the master node. The preprocessing step is also interleaved [81] at predefined intervals during processing, depending on which algorithm is used.

After preprocessing, the next step in both EIC and EOC algorithms is to compute the connected component structure of the graph. Both algorithms use an array-based union-find (also known as disjoint set) algorithm with path compression similar to that described in [82]. While typical union-find implementations employ a tree-based data structure, this implementation uses an array that is faster and is at least as memory efficient. The worst-case runtime is $O(|E|\alpha(|V|))$, and the average-case runtime is $O(|E|)$.

5.6.1 Fault Tolerance Design

Both algorithms are designed to handle hardware failures that are typical in the high-performance computing environment. The algorithms are implemented using a master/worker scheme where there is typically a single master node and multiple worker nodes. The master node keeps a log of the work assigned to each worker node. If a worker node becomes unresponsive, the master node simply removes the worker node from its list of available resources and then resubmits the unfinished work to another node. However, if the master node was to fail, then all work would be lost. Therefore, the worker node periodically checkpoints the current state of the anlysis. The master node can simply restart the analysis at the previous checkpoint in case of hardware or software failure.

5.6.2 Edge-in-Core Algorithm

The EIC algorith is deployed to solve the maximum clique problem on graphs that are too large to fit into core memory as a bit adjacency matrix, but is able to be stored in main memory using an adjacency list. Only a single pass over the external file is needed to input the entire graph into main memory. Given that the algorithm can store both the vertex list and the entire set of edges in memory at once, the EIC algorithm has access to edge information and is able to dissect the graph in an

efficient and intelligent manner. The basic concept of the EIC algorithm is illustrated in Figure 5.4 and pseudocode is presented in Algorithm 1.

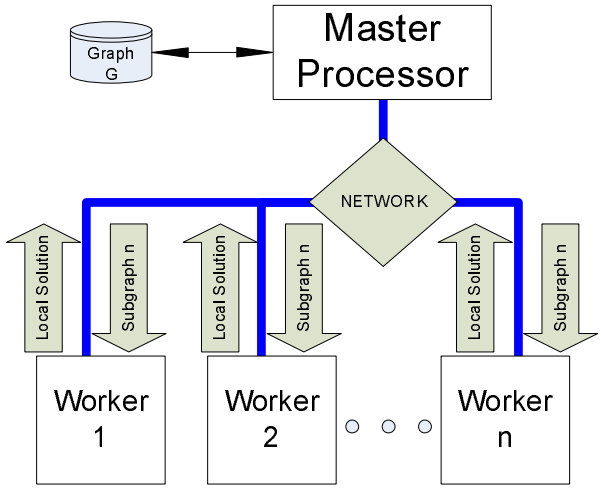


Figure 5.4: The EIC algorithm is a master/worker model, where only the master node needs disk access. The remaining data transfers are done via message passing.

Master Node - Edge-in-Core

The first step for the master node in the EIC algorithm is make an external pass over the file and read in the graph. The graph should be stored in a memory efficient data structure, such as an adjacency list. Next the graph undergoes preprocessing. Preprocessing consists of recursively removing all vertices with degree less than CMCS-1 or, in the case where CMCS is equal to 2, any vertex that has a degree of 1. The following step is to identifying the number of connected components. The connected component structure of the graph must yield one of the two following results: either all connected components have size at most M, or there is at least one connected component whose size exceeds M, which will be referred to as the large components. The former case is straightforward to process. The master node will pack as many connected components into a workload as possible and send to an available worker node. Each worker node then invokes MCF to compute the maximum clique size of the workload and returns the answer to the master node. The solution to

Input: Graph $G = (V, E)$
Output: Maximum clique size of G

Master Code
Read Graph G into memory and store as adjacency list
Run Preprocessing
Run Connected Component and Degree Structure
while *Unprocessed vertices exist* **do**
  foreach *Request for work from processor i* **do**
    Insert as many connected components (or neighborhoods of vertex r) as possible.
    Eliminate all possible vertices from search space
    Send bin to worker node i
    if *Worker node i returns a maximum clique size that exceeds CMCS* **then**
      Update CMCS
      Run Preprocessing
      Run Connected Components
    end
  end
  if *All workers have had at least one work segment or each connected component has had cut vertices removed* **then**
    Run Preprocessing
    Run Connected Components
  end
end

Worker Code
while *Available work* **do**
  Send request for work to master node
  foreach *Job received from master node* **do**
    Run Maximum Clique Solver on subgraph
    Send job results to master node
  end
end

Algorithm 1: The EIC algorithm is used for the case in which edge information is stored in core memory

maximum clique problem for the original graph is simply the largest maximum clique size of the connected components.

The second case, in which at least one connected component does not fit into main memory as a bit adjacency matrix, is not as straightforward. Like the previous case, EIC begins by sending all connected components small enough to fit in core to worker nodes. After these connected components are removed from the search space, the algorithm must extract subgraphs from the large connected component(s). These subgraphs are selected by extracting the induced neighborhoods of vertices. One important property that is exploited is the fact that solving the maximum clique problem on a subgraph that contains a vertex v and its neighborhood allows the algorithm to eliminate vertex v from the search space. Cut vertices in the connected component are selected first [83]. The master node adds the cut vertex, along with its induced neighborhood, to the bin to send to a worker node. Selecting a cut vertex of the large component guarantees that it will be split into two or more components, each one may or may not fit entirely into core memory as a bit adjacency matrix. If a bin is filled to capacity after adding the all cut vertices, it is sent to a worker node. Otherwise, subgraphs are extracted using the induced neighborhood of a root vertex r selected from one of the methods below:

- selecting a vertex of highest degree
- selecting a vertex of lowest degree
- selecting a random vertex

Selecting a vertex is contingent upon $|N_G(r)| < mB$, where mB is the size of available memory in the bin B. Once a root vertex has been selected, the search space is expanded by using a breadth-first search to find neighborhoods that either overlap or are disjoint from one another.

Regardless of the root vertex selection method, the approaches used to expand the search space are the same. Expanding the search space to include overlapping

neighborhoods begins by inserting the $N_G(r)$ into the bin along with every $N_G(w)$ where w is a neighbor of r and $|N_G(w)| < mB$. Note that if $N_G(w)$ is fully contained in $N_G(r)$ then both vertices w and r can be eliminated from the search space. Therefore, if the induced neighborhood of the root vertex is dense, then expanding the search space to include the induced neighborhoods of the low degree neighbors of the root vertex allows the algorithm to eliminate multiple vertices in a single step. On the other hand, if the induced neighborhood of the root vertex is sparse, then expanding the search space to include disjoint neighborhoods allows the aglorithm to eliminate at least one more vertex in addition to the root vertex. Disjoint neighborhoods are explored by first adding $N_G(r)$ into the bin and then performing a breadth-first search with the vertex r as the source node. Once a vertex w is discovered with a minimum distance of three from r, $N_G(w)$ is inserted into the bin contingent upon $|N_G(w)| < mB$. Therefore, at least one vertex is removed from the search space for each neighborhood selected. After the bin is filled to capacity or all remaining vertices in the search space are in the current bin, the bin containing the subgraph $G'$is sent to a worker node. After each worker gets at least one workload or each connected component has had all cut vertices removed, the master node interleaves the preprocessing step and recomputes the connected component structure.

Worker Node - Edge-in-Core

The worker nodes are able to receive the applicable workload from the master via message passing. After receiving a workload, the worker node will invoke a local instance of the MCF program to compute the maximum clique. The result generated by MCF is returned from the worker node to the master node and a new request for work is sent The master node will compare the result from the worker node against the current value of CMCS and update the value of CMCS accordingly. If the CMCS is updated, the master node immediately interleaves the preprocessing step, removing any vertices with degree less than CMCS-1. Following the completion of the preprocessing, the connected component structure of the graph is recomputed

and any worker nodes waiting for work are sent workloads based upon the reduced graph. The master node continues to process components until all vertices in G have been eliminated from the search space. Figure 5.5 illustrates the necessary hardware and overall workflow of the EOC algorithm.

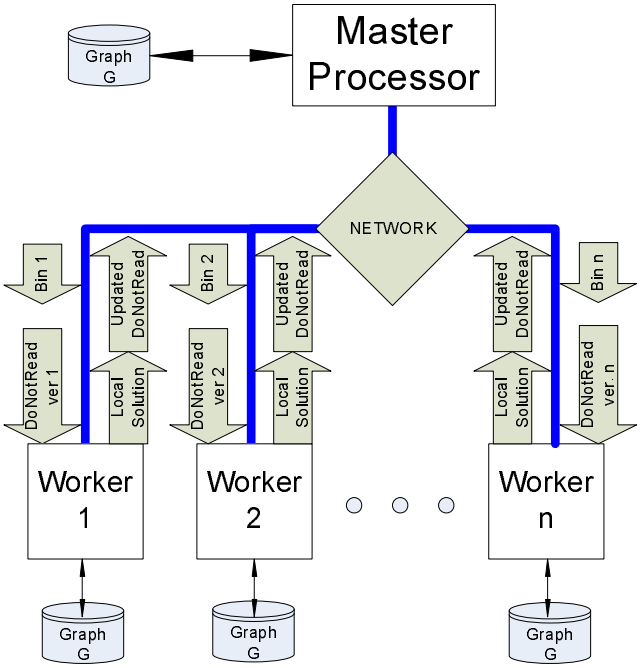


Figure 5.5: The EOC algorithm has the same basic master/worker setup as the EIC algorithm. Notice, however, the largest difference between these two algorithms is the need all the worker nodes to have access to the original graph on disk.

5.6.3 Edge-Out-of-Core Algorithm

The EOC algorithm is required for graphs for which the edge list will not fit in core memory, but the vertices and properties of the vertices and graph can be. Thus the algorithm must make multiple passes over the external file and recompute the connected components and degree structure on each pass. The degree structure is simply the degree of every vertex still in the search space. Since the edge list is not stored in core memory, the master node lacks the ability to build subgraphs based on neighborhoods of vertices. Note that unlike the EIC algorithm, where only the

master node needed access to the external file, the EOC algorithm requires every worker node, in addition to the master node, to have access to the external file. This requires each worker node to have either a local copy of the external file or needs to have access to a centrally shared file. The upside of having a local copy for each worker node is the ability to parallelize access to the external file across the N worker nodes and the master node. The downside of using local copies of the original file is the amount of external storage space needed to make N copies of the same data. Conversely, using a centrally located file, to which all nodes have access, requires only a single copy of the external file be stored on disk. However, access times for this file will not be nearly as efficient as having a local copy. In addition to the original external file, the master node also maintains a list of the vertices that have been removed from the search space, aptly named the do not read (DNR) list. This list allows any copy of the original graph to remain unchanged. Any edge that contains a vertex in the DNR list can be discarded when parsing the file. Before each workload is sent to the worker node, the current DNR list is sent to the worker node. Pseudocode for the EOC algorithm is presented in Algorithm 2.

Master Node - Edge-Out-of-Core

The underlying principles for the master node is similar for the EIC algorithm and the EOC algorithm. The master node begins its work by making an initial pass over the external file, only reading in valid edges with respect to the DNR list. During the initial pass, the DNR file is empty, thus all vertices and edges are read. Both the connected component and degree structures are computed during the first pass of the file. Unlike the EIC algorithm, the preprocessing for the EOC algorithm does not recursively remove vertices with degree less than CMCS-1. Only vertices that have degree less than CMCS-1 at the beginning of the preprocessing step are removed due to the fact that one pass over the external file is required for each recursive call. For large graphs, the external I/O time quickly becomes prohibitive. Therefore, in

Input: Graph $G = (V, E)$
Output: Maximum clique size of G

Master Code
while *Unprocessed vertices exist* **do**
  Read Graph G and get degree structure and connected components information
  foreach *Request for work from processor* i **do**
    Insert as many connected components (or neighborhoods of vertex r) as possible.
    Send DNR to processor i
    Send bin to processor i
    Add all possible vertex elements in bin to DNR
    if *Worker node i returns a clique size exceeding CMCS* **then**
      update CMCS
    end
    if *Worker node i returns an updated list of vertices to eliminate* **then**
      add set of vertices to DNR
    end
  end
end

Worker Code
while *Available work* **do**
  Send request for work to master node
  foreach *Job received from master node* **do**
    Read in subgraph $G^{'}$ from original graph file
    Run Maximum Clique Solver on subgraph of $G^{'}$
    Send job results to master node Send set of vertices to master node to eliminate
  end
end

Algorithm 2: The EOC algorithm is used for the case in which edge information will not fit into core memory.

order to keep the number of external passes over a file to a minimum, the connected components and degree structures are computed at predefined intervals.

Like the EIC algorithm, if the graph is constructed of multiple connected components, each with size at most M when stored as a bit adjacency matrix, then each worker node is assigned a workload comprised of connected components until every connected component is eliminated from the search space. Otherwise, if there is at least one connected component whose size exceeds M, then the master node must dissect this connected component into multiple, smaller components. It begins the dissection by selecting key vertices from these large components and building subgraphs around them.

Similarly to the EIC algorithm, different approaches may be used to select a root vertex to build a subgraph around. Unlike the EIC algorithm, however, it is not feasible to expand the search space around a selected vertex r in an efficient manner since edge information is not stored in core memory. Due to the lack of edge information in core memory, the neighborhood of vertex r must be read from the external file by the worker node, not the master node. Another difference is that only vertex r is placed in the bin to be sent to the worker node and not $N_G(r)$. The available space in the bin, mB, is still reduced by $|N_G(r)|$.

Any additional vertices that will be added to the bin will be selected from the same connected component from above. However, it is impossible to select a new vertex to add to the bin based on its connectivity to previously added vertices. For example, with respect to previously selected vertices, new candidates that will overlap with existing neighborhoods or are disjoint from existing neighborhoods cannot be guaranteed to be selected. This restricts the ability to fine tune the vertex selection mechanism due to the structure of the graph.

After the bin is filled to capacity, or all remaining vertices in the current search space are in the current bin, the vertices contained within the bin are added to the DNR list. Both the DNR list and the current workload are sent to the appropriate worker node. Once all workers have completed at least one workload, or if there

are multiple connected components, at least one neighborhood from each connected component has been processed, then the master node parses the external file to recompute the connected component and degree structures. The master node repeats this process until all vertices are eliminated from the search space.

Worker Node - Edge-Out-of-Core

The worker node in the EOC algorithm has a slightly more demanding role than it does in the EIC algorithm. The worker node must make a new pass over the external file each time a new workload is received, and it must also keep track of the DNR list. In the EOC algorithm, the worker node no longer receives the entire dataset from the master node via messages. Instead the worker node receives the DNR list along with a list of vertices that the worker node must build subgraphs around. The worker node must parse the external file, only reading in edges that contain at least one vertex present in the workers bin but not present in the DNR list.

The worker node must also keep track of the vertices that are entirely contained in the subgraphs that are being created. For example, if $N_G(r)$ is being processed and $N_G(w)$ is contained completely within $N_G(r)$, then vertex w can be added to the DNR list. It is the responsibility of the worker node to inform the master node of the set of vertices that were discovered in this manner and that they should be added to the global DNR list so they can be eliminated from the global search space. The worker node also returns the solution to the maximum clique problem to the master node and requests more work. After all of the vertices have been eliminated from the search space, the worker node will receive a broadcast message to finalize and exit.

5.7 Results

In order to demonstrate the scalability of the EIC and EOC algorithms, the Yeast dataset is once again analyzed. The tests were executed on a cluster of 32 nodes, each node containing two Intel Xeon 3.20 GHz processors with 4 GB of main memory and

connected via Myrinet. This is comparable to a typical setup of a compute cluster for a modest size lab. Given the modest size of this dataset, it is necessary to reduce the available memory to the bins, during the bin packing phase, in order to illustrate the scalability of the algorithms. Both algorithms were given the same input graph and the same memory restrictions. As seen in Figure 5.6, both algorithms scale well when increasing the size of the bins. Note that the minimum number of iterations over the graph is two, unless the entire graph fits into core memory, in which case there is no need to run the out-of-core algorithms as there is an impact in the running times associated with starting MPI processes.

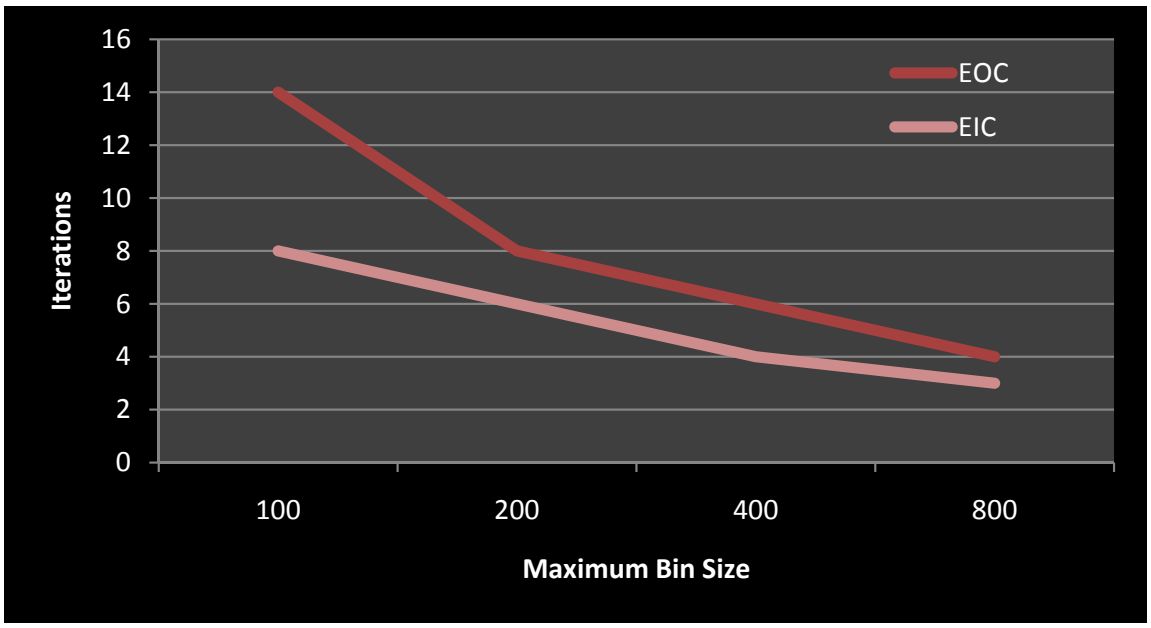


Figure 5.6: The effect of bin size for both the EIC and EOC algorithms are presented. Note that as the size of the bin increases, the number of iterations decreases.

Recall that for this particular graph the maximum clique size is 73 and that there are 27 distinct maximum cliques. All of the different experiments produced maximum cliques of size 73, however, different maximum cliques were produced. This is due to the fact that partitioning the graph into bins of different sizes results in different subgraphs. Given that both algorithms use local maximum clique sizes to prune

the remaining graph, the quicker an algorithm converges on a large clique, the more the algorithm can prune the search tree. This results in quicker convergence to the maximum clique size of the original graph. Figure 5.7 compares the relative speedup for both algorithms. In order to make fair comparisons, the maximum bin size was set to 100.

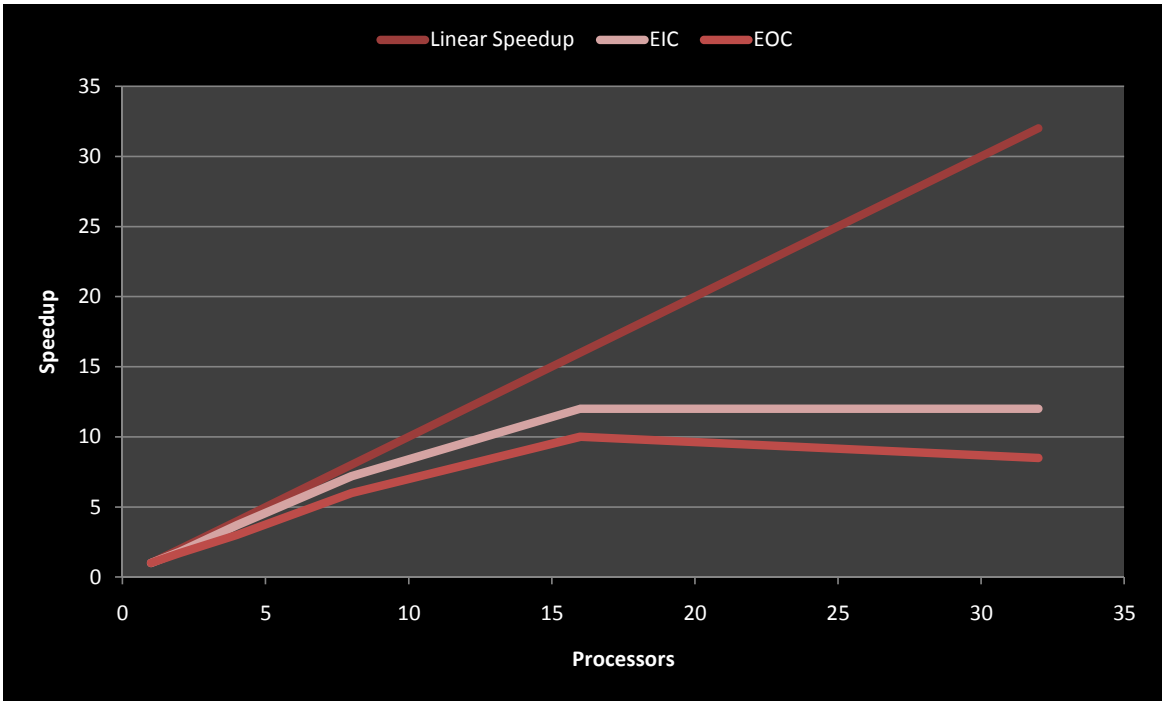


Figure 5.7: The relative speedup for both the EIC and EOC algorithms are presented. The algorithms start to diverge from linear speedup around 16 processors and begins to fall around 32 processors. This is due to the fact that the overhead associated with starting MPI processes begins to be significant with respect to the overall running times.

Although both algorithms do scale well, the results demonstrate that having the ability to select candidate vertices based upon connectivity properties, and not being required to make multiple passes external files, greatly improve the running times of out-of-core algorithms. Figure 5.8 breaks down the proportionality of time spent on tasks for each algorithm running on the master node. The most important factor is the time the EOC algorithm spends on disk access versus the EIC algorithm.

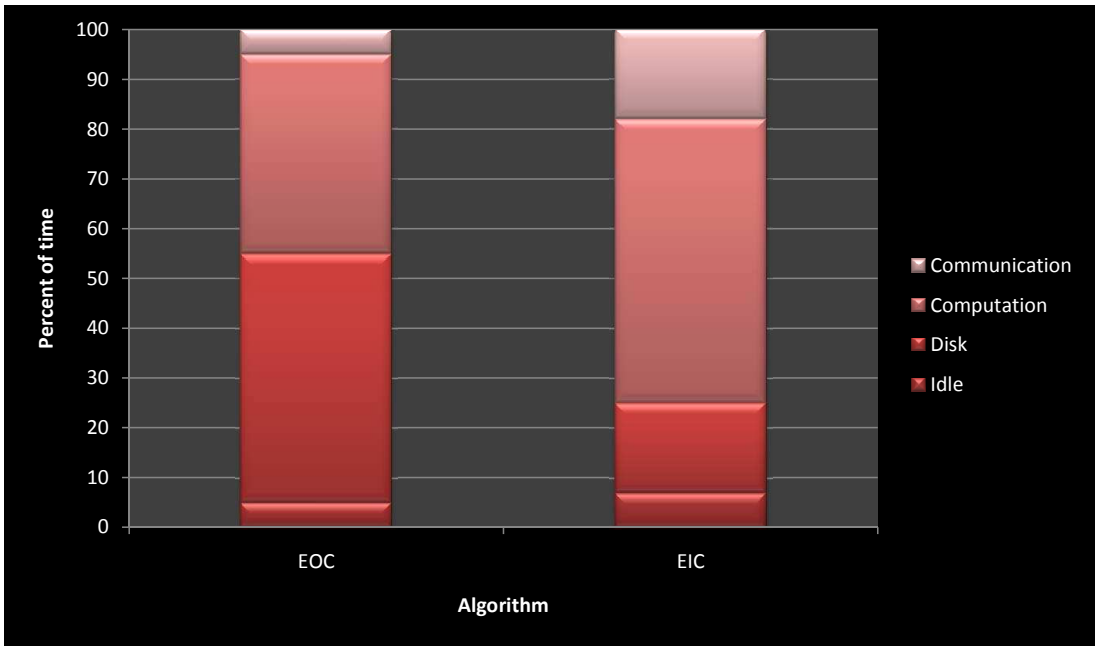


Figure 5.8: The EOC algorithm has the same basic master/worker setup as the EIC algorithm. Notice since the EOC algorithm cannot store edge information in memory, disk access takes up a significant portion of the running time.

In conclusion, these results show that the EIC algorithm is more efficient at solving the maximum clique problem than the EOC algorithm. However, it is important to note that the EOC algorithm can analyze much larger graphs than the EIC algorithm. The results of the experiments also show that both algorithms scale well with respect to memory requirements and they also scale well with the number of processors, relative to the difficulty of the problem. Given that no other algorithms exist to solve the maximum clique algorithm, it is not possible to compare the running time of these algorithms with anything else.

5.8 Applying Other Graph Based Algorithms to the Out-of-Core Framework

The EIC and EOC algorithms presented above can be extended to solve other graph algorithms such as paraclique. The only tweak needed to extend both of these

algorithms to solve paraclique lie in the method of splitting the graph into manageable segments. Where only the induced neighborhood of a root vertex is needed for the maximum clique algorithms, the induced neighborhood of the neighbors of the root vertex is needed for the paraclique algorithm. Another clique-centric algorithm that is trivially derived from the EIC and EOC algorithms is the anchored clique. An anchored clique approach can actually reduce the search space of an input graph since only the subgraphs that interact with the anchors need to be explored. However, it is feasible that these anchors are highly connected vertices in the graph and thus the entire graph would still have to be dissected into manageable segments in order to explore the entire search space.

5.9 Conclusion

This chapter reviewed two parallel algorithms aimed at solving the maximum clique problem on graphs that are too large to store in core memory. The first algorithm proposed, EIC, uses in-core techniques to dissect graphs into segments of manageable size. The second algorithm, EOC, requires multiple passes over external files for the master and worker nodes. Given the prohibitively expensive access to disk, this algorithm is deployed only on those graphs that are too large to exploit the in-core techniques used by the EIC algorithm.

5.10 Future Work

As the high-performance computing environment continues to change, so must the algorithms used on these systems. The algorithms above provide a solid foundation for future work of adapting and extending graph-based analyses to more advanced computing environments. These algorithms were designed and tested on machines that used commodity hardware such as standard hard drives, RAM, and network interfaces. Future work in exploring the analysis of large graphs might include

tweaking the above algorithms for use with special hardware such as solid-state drives, which would reduce the amount of time each algorithm would spend on disk I/O.

Chapter 6

Graph Algorithms Pipeline for Pathway Analysis

6.1 Introduction

The aforementioned algorithms are efficient at analyzing data, however, they lack an easy to use graphical user interface. The end user must download the source code, recompile the code for the machine architecture on which the code will run, and learn the applicable command line arguments for each of the tools. This is a daunting task for most users. Therefore, a new graphical interface toolkit is presented: the Graph Algorithms Pipeline for Pathway Analysis, GrAPPA. GrAPPA is an easy to navigate interface based on the Galaxy [84, 85] framework. The Galaxy framework allows for scientists and software developers to integrate data and software tools and it is becoming increasingly popular in the bioinformatics community. GrAPPA not only provides a graphical interface to the graph-based algorithms, but also provides computational resources to end users that might not have the appropriate resources readily available to them to deploy such analyses.

6.2 Motivation

In order for any software to be successful, it must be readily available to end users and it must be intuitive to use. GrAPPA accomplishes both of these requirements by providing an easy to use point-and-click environment accessible via the internet and gives even the most novice user the ability to upload data generated from microarrays and complete an entire analysis including preprocessing raw data, using the graph-based tools to analyze the data, and postprocessing using visualization. GrAPPA also allows for users to share datasets, workflows, and results. This enables other scientists easy access to run their own analysis and provides an easy method to reproduce the results of previous experiments.

6.3 Interface

GrAPPA's user interface is based on the basic design provided by the Galaxy framework, thus allowing users with familiarity of other tools based on the Galaxy framework to begin using GrAPPA without a steep learning curve. The basic interface is separated into four distinct regions, as seen in Figure 6.1. The topmost region is reserved for dataset management and user account information. The leftmost region is the location of all available tools. These tools are listed in a top-down order in which users should use them in the analysis of microarray data. The rightmost region of screen is the user's history. The history makes a record of the results at each step during the analysis and provides quick access to the data generated by each tool. Finally, the center area is working area. This is the main area and it is where the user completes such tasks as uploading raw microarray data, setting parameters for each of the different tools, and viewing the results of the analysis.

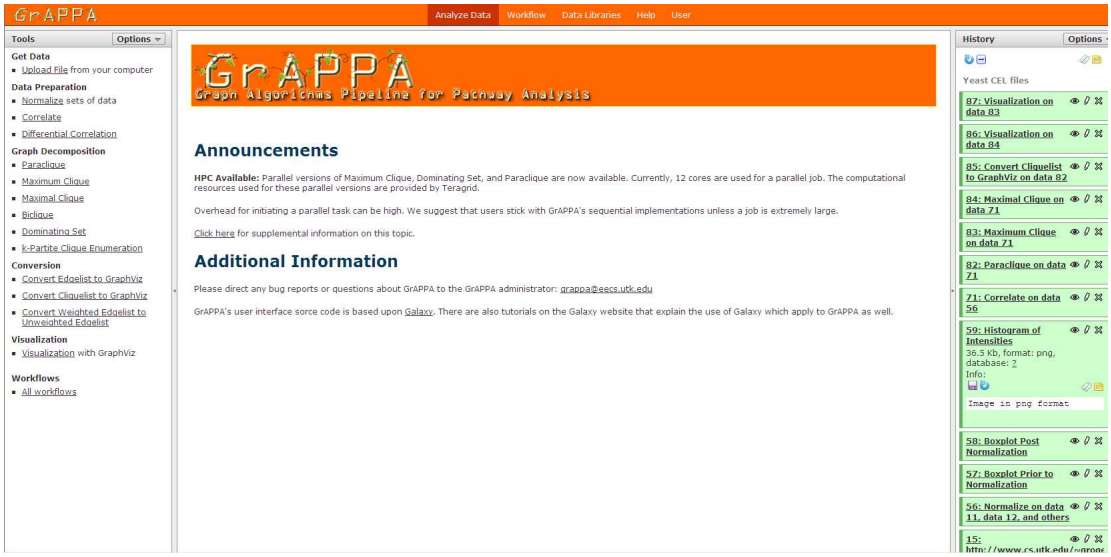


Figure 6.1: GrAPPA is implemented on the Galaxy framework. This enables users familiar with any tool based on the Galaxy framework to quickly navigate the GrAPPA interface.

6.3.1 Uploading Data

Data being analyzed by the GrAPPA tool can be submitted in one of two ways. First, the user can use the standard upload method, which allows data stored on the user's computer to be directly uploaded to the GrAPPA server. Secondly, data that is stored online can easily be retrieved by GrAPPA, avoiding the needless download and then upload step on the user's part. Once data has been uploaded to GrAPPA, the end user has a variety of options available to share the data. The most protected settings will allow access only to the user that uploaded the data. Data access can also be granted by the uploader of the data to any number of users in the GrAPPA database. Finally, data access can be granted to all users in the GrAPPA community. This restriction of data access grants users the privacy needed when first analyzing data, and then allows them to share the data and workflows with the general public once the results of the analysis are published. The types of data accepted by the GrAPPA toolkit range from raw microarray files (eg. Affymetrix CEL files) to DIMACS formatted graph files. The end user can chose to complete the entire analysis of microarray data

using GrAPPA or can input the correctly formatted data into any of the available tools anywhere in the analysis toolchain. This flexibility provides both novice and advanced users a quick and easy option to analyze their data.

6.3.2 Tools

The real substance behind the GrAPPA toolkit is the algorithms that are used behind the scenes. GrAPPA incorporates a mix of open-source software, such as R and GraphViz, as well as a large collection of custom software, such as the tools presented earlier in this dissertation. Incorporating this mix of different software modules gives GrAPPA the versatility to grow when new software is available to the bioinformatics community and to supplement existing tools when necessary. The tools currently available in GrAPPA are discussed in detail in the sections to follow.

6.3.3 Data Preparation

With respect to graph-based analysis, data preparation can be broken down into two distinct categories: data preprocessing and graph generation. Both of these categories were covered in detail in Chapter 3. GrAPPA provides access to the most popular algorithms available when preprocessing raw microarray data. For example, when preprocessing Affymetrix CEL files, the user has the option of using a wide variety of normalization methods, including RMA, MAS 5.0, and GCRMA. Options for generating boxplots of the data before and after normalization, and for generating a histogram of the log intensities of the data, are provided for a quick visualization of the data to determine the presence of any outliers or abnormalities. See Figure 6.2.

Generating a graph from the preprocessed microarray data can be a cumbersome task [38]. A plethora of different parameters in the generation of the graphs include the selection of a similarity metric and the selection of an appropriate threshold. GrAPPA enables the user to select from a list of the most popular similarity metrics and a user-defined threshold to generate the graph. If the user chooses to use a metric

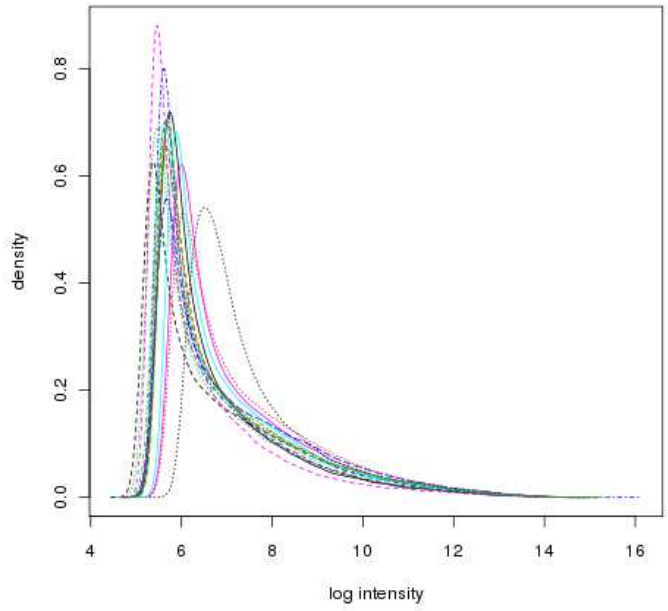


Figure 6.2: Visual outputs, such as this histogram, provide the user a snapshot of the data to quickly verify the existence of any corrupt data. The log intensities of the gene expressions derived from the Yeast data are plotted.

not listed, they can simply generate the graph offline using whichever criteria they would like and upload the final version of their graph in the proper format.

6.3.4 Graph Decomposition

Regardless if the graph was generated using the available GrAPPA tools or uploaded by the user, the graph can be analyzed using the available clique-centric tools to identify the underlying structures in the graph, namely the extremely dense regions. The tools in the graph decomposition section generally take a graph in DIMACS format as input, along with a set of parameters. Certain tools do not need input parameters, such as the Maximum Clique tool, while others have a large set of parameters that can be tweaked, such as the Paralique tool. GrAPPA restricts the list of input files to only those matching the proper format for the respective tool. This reduces the amount of user introduced errors while analyzing data. By default,

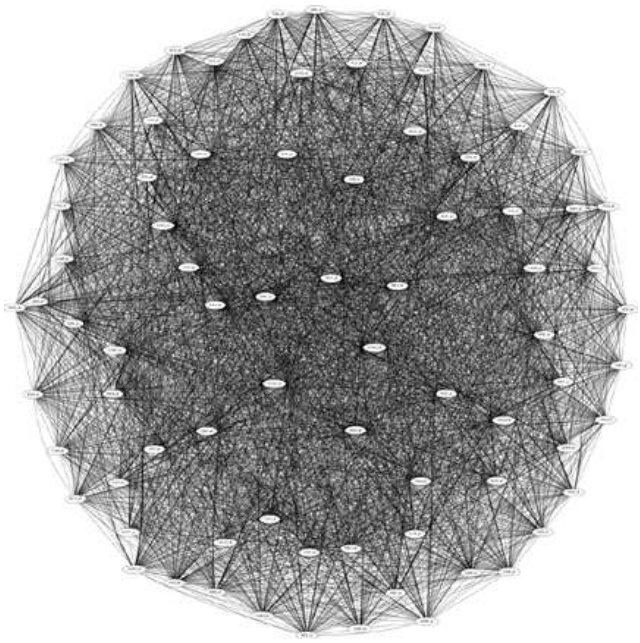
GrAPPA will set the parameter values to a generally acceptable level. Users have the ability to change these parameters, but are warned when selecting values that will effect the usability of the results. An example of this would include selecting a *glom* factor for the Paraclique tool that is larger than the size of the maximum clique. This would result in the inclusion of all vertices in the graph, thus rendering the analysis useless. Another example of unadvisable parameter selection would be to generate all maximal cliques in a large, dense graph. This would produce maximal clique results numbering in the billions, or even trillions, and would require large amounts of disk space to store.

6.3.5 Visualization

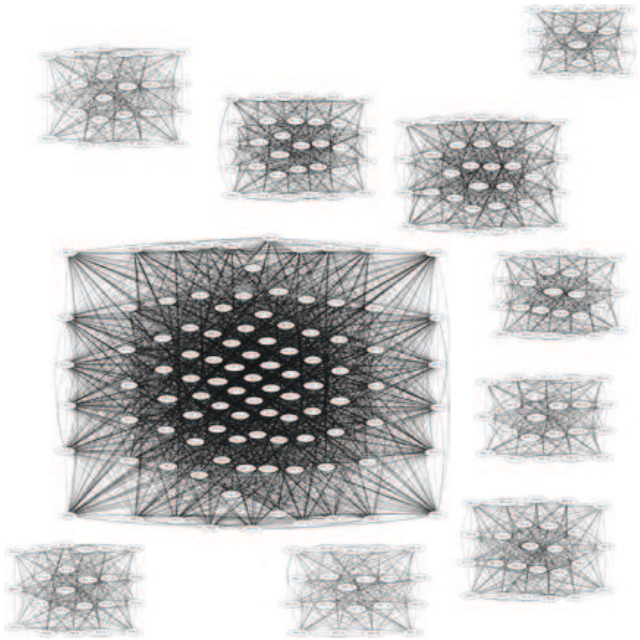
The visualization component of postprocessing provides a convenient way to interpret the structures in the resulting biological networks generated by the graph decomposition tools above. GrAPPA provides its visualization component using the GraphViz program. GraphViz provides a number of different graph layouts depending on the structure of the graph. For example, the *circo* layout method generates results where vertices and edges are in a circular pattern, while the *neato* layout method uses a spring based model to place the nodes and edges. See Figure 6.3.

6.3.6 History

One of the most useful features in GrAPPA is the ability to track the history of an analysis. Each step in the analysis is recorded, including the input, parameters, and the result. This enables a user to rerun a portion of the same analysis quickly at a later date, or to tweak a previous analysis when only one or two parameters needs to be adjusted. A user's history also provides a methodology for others to reproduce any result generated using GrAPPA. Similarly to the ability to share datasets, mentioned above, a user can chose to not allow anyone else to access their history or to share



(a)



(b)

Figure 6.3: GraphViz is used to generate visual representations of the resulting networks. The overlapping maximal cliques generated from the Yeast data can be seen in (a). The non-overlapping paraclique results on the same graph can be seen in (b).

their history with other users. A single history is typically associated with a single analysis, although a user can save multiple histories.

6.3.7 Workflow

The workflow in GrAPPA provides a visualization of the history as well as a method to rerun the same analysis, using the same parameters, but with different data. The visualization aspect enables the user to quickly identify the input and output of every step in the analysis along with the overall flow of data from the preprocessing of microarray data to visualization of the results generated by the clique-centric tools. The ability to rerun the same analysis on different input, while not having to set the parameters, allows the user to compare results from two separate inputs in a quick and efficient manner. An example workflow is seen in Figure 6.4. The workflow and history are deeply related. Workflows can be automatically generated from a history, and both provide the user with the ability to rerun previous analyses.

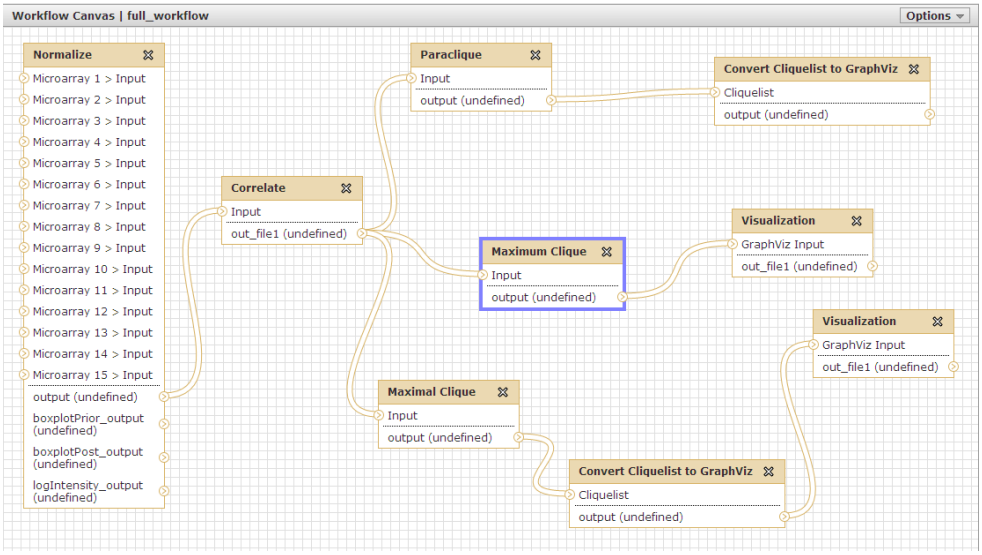


Figure 6.4: Workflows in GrAPPA allow for the same analysis to be quickly recomputed. It is also a way to visually design the analysis, from data input to postprocessing.

6.4 Computational Resources

In addition to providing an intuitive interface to analyze microarray data, GrAPPA also provides the computational resources needed to run the tools that exploit the combinatorial algorithms. GrAPPA is currently hosted on a server with 8 cores running at 2.40 GHz with 12 GB of DDR3 RAM. This machine serves as both the web interface and the primary computational resource. All preprocessing and postprocessing occurs on this machine, as well as any serially enabled job instance of the tools listed above. If a job requires a large computational component, then GrAPPA will deploy parallel versions of the algorithms and farm the work an available machine on the TeraGrid [86]. GrAPPA currently has access to four HPC machines including Lonestar at Texas Advanced Computing Center (TACC), Abe at National Center for Supercomputing Applications (NCSA), Steele at Purdue University, and Queen Bee at Louisiana Optical Network Initiative (LONI). The machines have a computational peak performance range from 50.7 Petaflops at Queen Bee to 302 Petaflops at Lonestar. GrAPPA acts as a science gateway for these TereGrid resources, which allows researchers without access to HPC machines to analyze large datasets which otherwise would be impossible. A diagram of the typical workflow of an analysis on GrAPPA is illustrated in Figure 6.5. Note that the choice of TeraGrid machines are not the largest available supercomputers on the TeraGrid network. The selected machines were based on a wide variety of factors such as uptime, availability, and utilization. These machines provide the end user a good balance between computational power and quick response time. GrAPPA strives to provide an environment that is as near to real-time as possible, therefore scheduling parallel jobs on compute resources that are heavily utilized could result in long queue times.

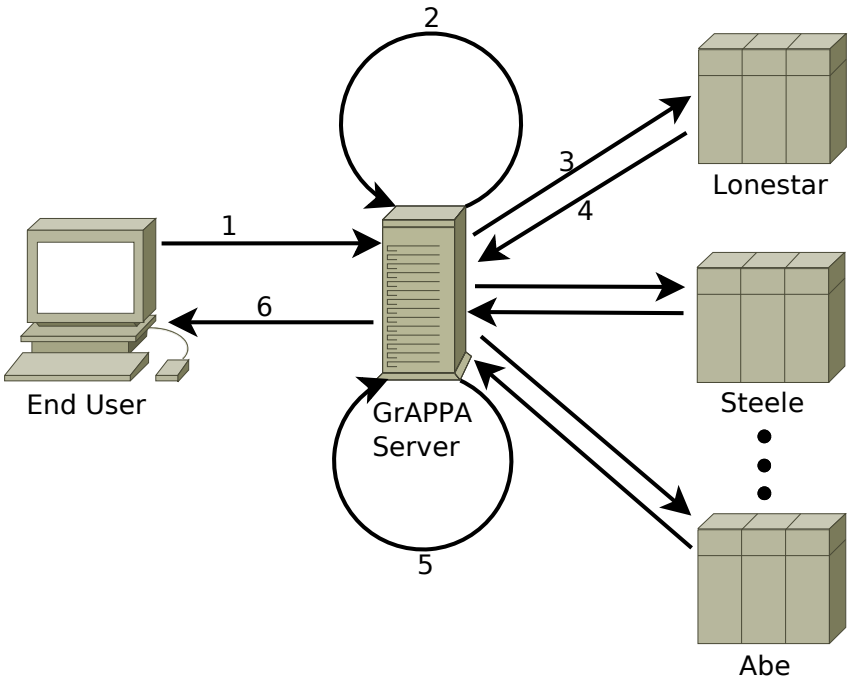


Figure 6.5: A researcher uploads raw microarray data to GrAPPA via the internet(1). The preprocessing of the data and any serial jobs are ran on GrAPPA(2). The computationally difficult jobs are offloaded to one of the many HPC resources available via the TeraGrid(3). The jobs are queued at the HPC center and results are returned to GrAPPA once completed(4). The postprocessing, including visualization, is completed on the GrAPPA server(5). Final results of the analysis are returned to the user(6). The only hardware resource required of the end user is a computer connected to the internet. All other hardware and software requirements are provided by GrAPPA.

6.5 Future Work

GrAPPA currently provides an easy to use graphical interface for members in the bioinformatics community to efficiently analyze microarray data using graph-based methods. Future work in GrAPPA could include growing its user base by providing support for other popular network based methods, such as Bayesian networks. Extending GrAPPA to interact with other Galaxy based tools would provide users with many different analytical tools. Finally, GrAPPA could grow support other biological data sources such as Next Generation Sequencing. Parallel tools are currently coming online to process this type of data, and the amount of computational

power and storage area needed to analyze this type of data would be a perfect fit for a tool such as GrAPPA.

Chapter 7

Conclusions

7.1 Review

Different methods of analyzing high-throughput microarray data using clique-centric tools were presented in this dissertation. The first step in this process was to transform the graph from raw data, generated by a collection microarray chips, to a simple, finite, undirected graph. This process is the summation of three distinct subprocesses. The first subprocess is the normalization of the raw data. This is necessary to correct for effects of variation in the microarray technology rather than true biological variation between the samples. A variety of normalization methods were tested using a *Saccharomyces cerevisiae* dataset to illustrate the different expression values produced. The data is then subjected to the second subprocess, which is the computation of all pairwise correlations between the probes on the chips. The data generated by each of the normalization methods was the input for each of the similarity metrics studied. The combination of both the normalization method and the similarity metric produced a complete graph, where the vertices represented the probes on the chip and the edges were weighted using the applicable similarity value. The third subprocess in the graph transformation was the threshold selection, which generates a graph that retains only putatively biologically-significant edges. Different

threshold selection methods surveyed, ranging from the use of statistical significant correlations to the value at which underlying graph structures doubled or tripled in size. In total, there were six normalization methods, five similarity metrics, and six thresholding procedures examined. All possible combinations of the aforementioned methods were used to generate a total of 180 graphs. These graphs were scrutinized for differences in size, density, and connectivity. It is concluded that it is simply not the case that one method from each of the three transformation steps always produces the best results. A wide range of these methods should be examined for each high-throughput microarray analysis in order to determine which collection of the methods is most applicable for the data being analyzed.

A subset of the *Saccharomyces cerevisiae* graphs were examined for their underlying graph structures. A suite of clique-centric tools were used to extract dense clusters, such as maximum cliques, from the graphs. Maximal clique profiles were also analyzed, along with the size and density of paracliques. Methods for tuning the Paraclique algorithm to generate results based on parameters such as maximum density and maximum edge weight were reviewed. A suite of tools based on a priori interactions between a set of genes was presented and applied to a dataset generated from *Mus musculus*. The underlying structures of all graphs examined varied widely depending on the methods used to generate the graphs.

Two algorithms were introduced that expanded the suite of clique-centric algorithms to use in-core and out-of-core techniques in order to analyze graphs that were simply too large to store in core memory. These algorithms used a distributed-memory programming model and the structure of the graph to solve local instances of the global problem. Pitfalls in this type of analysis, such as the required use of external memory, were examined. These algorithms were tested on two large graphs. The smaller of the two graphs was able to exploit the use of in-core techniques in the analysis, while the second graph was relegated to using strictly out-of-core techniques. The algorithms successfully analyzed both graphs and generated the size of the maximum clique in each.

Finally, the web-base tool GrAPPA was introduced. GrAPPA enables researchers to analyze microarray data without having to deal with installing software or setting up hardware. This results in more real analysis being completed by researchers. The graphical user interface is a point-and-click environment where researchers can upload raw microarray data and extract dense putative biological networks. GrAPPA also provides the same researchers access to high-performance computers on which to run the most computationally difficult algorithms in the graph-based toolchain.

7.2 Future work

The work presented in this dissertation illustrates that the methods used to transform biological data into a graph do not always generate the same type of graphs. Future work in this area would include research in different normalization methods, similarity metrics, and thresholding tools to determine if the resulting graphs provide better results than the current methods. Also, the analysis focused primarily on graphs generated from high-throughput microarrays measuring gene expression of different eukaryotes. Extending this analysis to other biological data would be of extreme interest.

The use of the out-of-core methods can be supplemented by the use of faster implementations of the underlying clique-centric tools, more memory-efficient data structures, and the exploitation of new technologies that reduce the amount of time spent on disk I/O. Different hardware designs will require tweaks to the current algorithms. The shift towards high-performance hybrid systems, that include both CPUs and GPUs, will require a new approach to these algorithms.

Lastly, GrAPPA can be extended to incorporate any new graph-based analysis tools tools. This will ensure that GrAPPA is a successful bioinformatics tool for years to come. GrAPPA can also be integrated with other Galaxy based tools in order to offer the non-GrAPPA users an opportunity to analyze their data using efficient graph-based network analysis.

Bibliography

Bibliography

[1] B. H. Voy, J. A. Scharff, A. D. Perkins, A. M. Saxton, B. Borate, E. J. Chesler, L. K. Branstetter, and M. A. Langston, "Extracting gene networks for low-dose radiation using graph theoretical algorithms," *PLoS Computational Biology*, vol. 2, p. 89, 2006. 2, 10, 12, 18

[2] F. N. Abu-khzam, N. E. Baldwin, M. A. Langston, and N. F. Samatova, "On the relative efficiency of maximal clique enumeration algorithms, with application to high-throughput," in *Computational Biology, Proceedings, International Conference on Research Trends in Science and Technology*, 2005. 2, 10

[3] N. E. Baldwin, E. J. Chesler, S. Kirov, M. A. Langston, J. R. Snoddy, R. W. Williams, and B. Zhang, "Computational, integrative, and comparative methods for the elucidation of genetic coexpression networks," *Journal of Biomedicine and Biotechnology*, vol. 2, pp. 172–180, 2005. 2, 10, 18

[4] R. M. Lynch, S. Naswa, G. L. Rogers, S. A. Kania, S. Das, E. J. Chesler, A. M. Saxton, M. A. Langston, and B. H. Voy, "Identifying genetic loci and spleen gene coexpression networks underlying immunophenotypes in bxd recombinant inbred mice," *Physiological Genomics*, vol. 41, no. 3, pp. 244–253, 2010. 2, 78, 79, 82

[5] J. D. Eblen, I. C. Gerling, A. M. Saxton, J. Wu, J. R. Snoddy, and M. A. Langston, "Graph algorithms for integrated biological analysis, with applications

to type 1 diabetes data," in *Clustering Challenges in Biological Networks*, pp. 207–222, World Scientific, 2008. 2, 18

[6] 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 *In Proceedings, Supercomputing*, p. 12, 2005. 2, 18, 64

[7] M. A. Langston, A. D. Perkins, A. M. Saxton, J. A. Scharff, Brynn, and H. Voy, "Innovative computational methods for transcriptomic data analysis," in *In Proceedings, ACM Symposium on Applied Computing*, pp. 190–194, ACM Press, 2006. 2

[8] C. Lee and M. Roy, "Analysis of alternative splicing with microarrays: successes and challenges," *Genome Biology*, vol. 5, 2004. 2

[9] M. K. Kerr and G. A. Churchill, "Experimental design for gene expression microarrays," *Biostatistics*, vol. 2, no. 2, pp. 183–201, 2001. 2, 7

[10] Q. Tan, K. Brusgaard, T. A. Kruse, E. Oakeley, B. Hemmings, H. Beck-Nielsen, L. Hansen, and M. Gaster, "Correspondence analysis of microarray time-course data in case-control design," *Journal of Biomedical Informatics*, vol. 37, pp. 358–365, October 2004. 2

[11] G. K. Smyth, J. Michaud, and H. S. Scott, "Use of within-array replicate spots for assessing differential expression in microarray experiments," *Bioinformatics*, vol. 21, no. 9, pp. 2067–2075. 2

[12] M. Sirota, M. A. Schaub, S. Batzoglou, W. H. Robinson, and A. J. Butte, "Autoimmune disease classification by inverse association with snp alleles," *PLoS Genet*, vol. 5, p. e1000792, 12 2009. 3

[13] M. Schena, D. Shalon, R. W. Davis, and P. O. Brown, "Quantitative monitoring of gene expression patterns with a complementary dna microarray," *Science*, vol. 270, no. 5235, pp. 467–470, 1995. 3

[14] T. Barrett, D. B. Troup, S. E. Wilhite, P. Ledoux, C. Evangelista, I. F. Kim, M. Tomashevsky, K. A. Marshall, K. H. Phillippy, P. M. Sherman, R. N. Muertter, M. Holko, O. Ayanbule, A. Yefanov, and A. Soboleva, "Ncbi geo: archive for functional genomics data sets10 years on," *Nucleic Acids Research*, vol. 39, no. suppl 1, pp. D1005–D1010, 2011. 3

[15] M. Barnes, J. Freudenberg, S. Thompson, B. Aronow, and P. Pavlidis, "Experimental comparison and cross-validation of the Affymetrix and Illumina gene expression analysis platforms," *Nucleic Acids Research*, vol. 33, no. 18, pp. 5914–5923. 3

[16] http://www.affymetrix.com. 4

[17] http://www.illumnia.com. 4

[18] B. M. Bolstad, R. A. Irizarry, M. Astrand, and T. P. Speed, "A comparison of normalization methods for high density oligonucleotide array data based on variance and bias," *Bioinformatics*, vol. 19, pp. 185–193, Jan 2003. 5, 28

[19] R. A. Irizarry, B. M. Bolstad, F. Collin, L. M. Cope, B. Hobbs, and T. P. Speed, "Summaries of Affymetrix GeneChip probe level data," *Nucleic Acids Res*, vol. 31, p. e15, Feb 2003. 5

[20] M. N. McCall and R. A. Irizarry, "Consolidated strategy for the analysis of microarray spike-in data," *Nucleic Acids Research*, vol. 36, no. 17, p. e108, 2008. 5

[21] E. C. Rouchka, A. W. Phatak, and A. v. Singh, "Effect of single nucleotide polymorphisms on affymetrix match-mismatch probe pairs," *Bioinformation*, vol. 2, pp. 405–411, 2008. 5

[22] M. Consortium, "The microarray quality control (maqc) project shows inter- and intraplatform reproducibility of gene expression measurements," *Nature Biotechnology*, vol. 24, pp. 1151–1161, 2006. 5, 6

[23] N. L. Barbosa-Morais, M. J. Dunning, S. A. Samarajiwa, J. F. J. Darot, M. E. Ritchie, A. G. Lynch, and S. Tavar, "A re-annotation pipeline for Illumina BeadArrays: improving the interpretation of gene expression data," *Nucleic Acids Research*, vol. 38, no. 3, p. e17, 2010. 6

[24] D. Abdueva, M. R. Wing, B. Schaub, and T. J. Triche, "Experimental comparison and evaluation of the affymetrix exon and u133plus2 genechip arrays," *PLoS ONE*, vol. 2, no. 9, p. e913, 2007. 6

[25] L. D. Stein, "Human genome: End of the beginning," *Nature*, vol. 431, pp. 915–916, 2004. 6

[26] R Development Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0. 8, 27

[27] R. C. Gentleman, V. J. Carey, D. M. Bates, *et al.*, "Bioconductor: Open software development for computational biology and bioinformatics," *Genome Biology*, vol. 5, p. R80, 2004. 8, 9, 27

[28] Gnome Qubec, Montreal, Canada, *FlexArray: A statistical data analysis software for gene expression microarrays.*, 2007. 8, 29

[29] G. K. Smyth and T. Speed, "Normalization of cdna microarray data," *Methods*, vol. 31, no. 4, pp. 265 – 273, 2003. Candidate Genes from DNA Array Screens: application to neuroscience. 9

[30] R. A. Irizarry, B. Hobbs, F. Collin, Y. D. Beazer-Barclay, K. J. Antonellis, U. Scherf, and T. P. Speed, "Exploration, normalization, and summaries of high

density oligonucleot ide array probe level data," *Biostat*, vol. 4, no. 2, pp. 249–264, 2003. 9, 29

[31] C. Li and W. H. Wong, "Model-based analysis of oligonucleotide arrays: Expression index computation and outlier detection," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 91, no. 1, pp. 31–36, 2001. 9

[32] F. Millenaar, J. Okyere, S. May, M. van Zanten, L. Voesenek, and A. Peeters, "How to decide? different methods of calculating gene expression from short oligonucleotide array data will give different results," *BMC Bioinformatics*, vol. 7, no. 1, p. 137, 2006. 9

[33] X. Cui, J. T. G. Hwang, J. Qiu, N. J. Blades, G. A. Churchill, and G. A. Churchill, "Improved statistical tests for differential gene expression by shrinking variance components estimates," *Biostatistics*, vol. 6, pp. 59–75, 2005. 9

[34] S. Dudoit, Y. H. Yang, M. J. Callow, and T. P. Speed, "Statistical methods for identifying differentially expressed genes in replicated cdna microarray experiments," 2002. 9

[35] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 57, no. 1, pp. pp. 289–300, 1995. 9

[36] A. Dabney, J. D. Storey, and with assistance from Gregory R. Warnes, *qvalue: Q-value estimation for false discovery rate control*, 2009. R package version 1.20.0. 10, 27

[37] B. Borate, "Comparative analysis of thresholding algorithms for microarray-derived gene correlation matrices.," master's thesis, University of Tennessee, 2008. 11, 53, 54

[38] A. D. Perkins, *Addressing Challenges in a Graph-Based Analysis of High-Throughput Biological Data*. Phd dissertation, University of Tennessee, 2008. 11, 53, 58, 64, 111

[39] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979. 11, 20

[40] T. F. Gonzalez, *Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series)*. Chapman & Hall/CRC, 2007. 11

[41] R. G. Downey and M. R. Fellows, *Parameterized Complexity*. Springer, 1999. 11

[42] International HapMap Consortium 12

[43] http://www.graphviz.org. 13

[44] http://www.cyotscape.com. 13

[45] http://www.david.abcc.ncifcrf.gov. 13

[46] http://www.geneontology.org. 13

[47] http://www.ingenuity.com. 13

[48] R. M. Karp, "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations* (R. E. Miller and J. W. Thatcher, eds.), pp. 85–103, Plenum Press, 1972. 19, 22

[49] J. D. Eblen, *The Maximum Clique Problem: Algorithms, Applications, and Implementations*. Phd dissertation, University of Tennessee, 2010. 20, 23, 64, 83, 90

[50] R. Niedermeier, *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, USA, 2006. 20, 21, 22

[51] J. F. Buss and J. Goldsmith, "Nondeterminism within p," *SIAM J. Comput.*, vol. 22, pp. 560–572, 1993. 21

[52] J. W. Moon and L. Moser, "On cliques in graphs," *Israel Journal of Mathematics*, vol. 3, pp. 23–28, 1965. 23

[53] F. Kose, W. Weckwerth, T. Linke, and O. Fiehn, "Visualizing plant metabolomic correlation networks using clique-metabolite matrices," *Bioinformatics*, vol. 17, pp. 1198–1208, 2001. 23

[54] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, 1973. 23, 77

[55] J. D. Eblen, C. A. Phillips, G. L. R. Jr., and M. A. Langston, "The maximum clique enumeration problem: Algorithms, applications and implementations." Unpublished manuscript, 2010. 24, 65, 73, 74, 77

[56] J.-P. Pitknen, A. Trm, S. Alff, L. Huopaniemi, P. Mattila, and R. Renkonen, "Excess mannose limits the growth of phosphomannose isomerase pmi40 deletion strain of saccharomyces cerevisiae," *Journal of Biological Chemistry*, vol. 279, no. 53, pp. 55737–55743, 2004. 27

[57] R. A. Irizarry, Z. Wu, and H. A. Jaffee, "Comparison of Affymetrix GeneChip expression measures.," *Bioinformatics (Oxford, England)*, vol. 22, no. 7, pp. 789–794, 2006. 28

[58] M. J. Korenberg, ed., *Microarray Data Analysis: Methods and Applications*. Totowa, NJ, USA: Humana Press, 2007. 29

[59] Z. Wu, R. A. Irizarry, R. Gentleman, F. Martinez-Murillo, and F. Spencer, "A Model-Based Background Adjustment for Oligonucleotide Expression Arrays," *Journal of the American Statistical Association*, vol. 99, no. 468, pp. 909+. 29

[60] J. Pevsner, *Bioinformatics and functional genomics*. Wiley-Blackwell, 2009. 33

[61] J. Quackenbush, "Microarray data normalization and transformation," *Nature Genetics*, vol. 32 Suppl, pp. 496–501, Dec. 2002. 35

[62] A. Sturn, J. Quackenbush, and Z. Trajanoski, "Genesis: cluster analysis of microarray data," *Bioinformatics*, vol. 18, no. 1, pp. 207–208, 2002. 35

[63] A. J. Butte, I. S. Kohane, and I. S. Kohane, "Mutual information relevance networks: Functional genomic clustering using pairwise entropy measurements," *Pacific Symposium on Biocomputing*, vol. 5, pp. 415–426, 2000. 35

[64] J. D. Storey, "A direct approach to false discovery rates," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 64, no. 3, pp. 479–498, 2002. 50, 51

[65] S. R. Narum, "Beyond bonferroni: Less conservative analyses for conservation genetics," *Conservation Genetics*, vol. 7, no. 5, pp. 783–787, 2006. 51

[66] A. Reiner, D. Yekutieli, and Y. Benjamini, "Identifying differentially expressed genes using false discovery rate controlling procedures," *Bioinformatics*, vol. 19, no. 3, pp. 368–375, 2003. 51

[67] J. D. Storey and R. Tibshirani, "Statistical significance for genome-wide experiments," *Proceedings of the National Academy of Sciences*, vol. 100, pp. 9440–9445, 2003. 51

[68] I. Hedenfalk, D. Duggan, Y. Chen, M. Radmacher, M. Bittner, R. Simon, P. Meltzer, B. Gusterson, M. Esteller, M. Raffeld, Z. Yakhini, A. Ben-Dor, E. Dougherty, J. Kononen, L. Bubendorf, W. Fehrle, S. Pittaluga, S. Gruvberger,

N. Loman, O. Johannsson, H. Olsson, B. Wilfond, G. Sauter, O.-P. Kallioniemi, k. Borg, and J. Trent, "Gene-expression profiles in hereditary breast cancer," *New England Journal of Medicine*, vol. 344, no. 8, pp. 539–548, 2001. 52

[69] E. J. Chesler and M. A. Langston, "Combinatorial genetic regulatory network analysis tools for high throughput transcriptomic data," in *Proceedings, RECOMB Satellite Workshop on Systems Biology and Regulatory Genomics*, pp. 150–165, 2005. 67

[70] C. A. Phillips, "Personal communication.," 2008. 71

[71] P. Du, W. A. Kibbe, and S. M. Lin, "lumi: a pipeline for processing Illumina microarray," *Bioinformatics*, vol. 24, no. 13, pp. 1547–1548, 2008. 78

[72] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you?," 2007. 87

[73] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 28th ACM symposium on Principles of distributed computing*, PODC '09, (New York, NY, USA), pp. 6–6, ACM, 2009. 88

[74] "Pegasus: A peta-scale graph mining system - implementation and observations.," 2009. 88

[75] J. Abello, A. L. Buchsbaum, and J. R. Westbrook, "A functional approach to external graph algorithms," in *Algorithmica*, pp. 332–343, Springer-Verlag, 1998. 90

[76] J. Abello, P. M. Pardalos, and M. G. C. Resende, "On maximum clique problems in very large graphs," in *In External Memory Algorithms*, pp. 119–130, American Mathematical Society, 1999. 90

[77] F. N. Abu-khzam, M. A. Langston, P. Shanbhag, and C. T. Symons, "Scalable parallel algorithms for fpt problems," tech. rep., Algorithmica, 2006. 90

[78] P. M. Pardalos, J. Rappe, Mauricio, and M. G. Resende, "An exact parallel algorithm for the maximum clique problem," in *In High Performance and Software in Nonlinear Optimization*, pp. 279–300, Kluwer Academic Publishers, 1998. 90

[79] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Computing*, vol. 22, pp. 789–828, Sept. 1996. 91

[80] W. D. Gropp and E. Lusk, *User's Guide for* `mpich`*, a Portable Implementation of MPI.* Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6. 91

[81] R. Niedermeier and P. Rossmanith, "A general method to speed up fixed-parameter-tractable algorithms," 1999. 92

[82] K. Wu and E. Otoo, "Lbnl-57527 a simpler proof of the average case complexity of union-find with path compression," 2005. 93

[83] J. E. Hopcroft and R. E. Tarjan, "Efficient algorithms for graph manipulation," tech. rep., Stanford, CA, USA, 1971. 96

[84] J. Goecks, A. Nekrutenko, J. Taylor, and T. G. Team, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.," *Genome Biology*, vol. 11, no. 8, 2010. 108

[85] D. Blankenberg, G. Von Kuster, N. Coraor, G. Ananda, R. Lazarus, M. Mangan, A. Nekrutenko, and T. J, *Galaxy: a web-based genome analysis tool for experimentalists.* 2010. 108

[86] http://www.teragrid.org. 116

Vita

Gary L. Rogers Jr. was born in San Francisco in 1981. He graduated from Cumberland County High School in Crossville, TN in 1999. In 2003, he received his BS in Computer Science from the University of the South in Sewanee, TN. He moved to Knoxville, TN where he received his MS in Computer Science from the University of Tennessee in 2005. He worked under the direction of Dr. Michael Langston at the Langston Lab and earned his PhD in Computer Science in 2011.