2-24-2020

Clustering methods for gene expression data of *Oxytricha trifallax*

Kyle Houfek
University of South Florida

Clustering methods for gene expression data of *Oxytricha trifallax*

by

Kyle Houfek

A thesis submitted in partial fulfillment
of the requirements for the degree of
M.A. in Mathematics
Department of Mathematics & Statistics
College of Arts and Sciences
University of South Florida

Major Professor: Natasha Jonoska, Ph.D.
Theodore Molla, Ph.D.
Masahiko Saito, Ph.D.

Date of Approval:
March 11, 2020

Keywords: Ciliates, Topology, Hierarchical, K-means, Gene Ontology

Table of Contents

List of Figures

Clustering methods for gene expression data of *Oxytricha trifallax*

Kyle Houfek

ABSTRACT

Clustering is a data analysis method which is used in a large variety of research fields. Many different algorithms exist for clustering, and none of them can be considered universally better than the others. Different methods of clustering are expounded upon, including hierarchical clustering and k-means clustering. Topological data analysis is also described, showing how topology can be used to infer structural information about the data set. We discuss how one finds the validity of clusters, as well as an optimal clustering method, and conclude with how we used various clustering methods to analyze transcriptome data from the ciliate *Oxytricha trifallax*. We discuss the structure of the data set, how an optimal clustering was chosen for this data set, how the validity of the clusters was confirmed, and how biological information can be extracted using gene ontology.

Chapter 1
Introduction

Our goal of this project is to use a data analysis method known as clustering to identify correlations among expression rates of genes, and to use these correlations to better understand biological processes which are happening in a cell at different times. This involves measuring transcriptome levels at certain time points across a given time frame, then using these readings to determine which genes had very high expression rates at the same time points. In this paper, we analyze multiple clustering methods, and discuss the advantages and disadvantages of each when dealing with different data sets. We also discuss how clustering applies to a specific genomic data.

The data set obtained from the Landweber Lab at Columbia University measures the transcriptome levels of the ciliate organism known as *Oxytricha trifallax*, taking readings at 12 hour intervals across 72 hours, for a total of six readings. This data set was normalized using a DeSeq 2 package in R [17], the details of which will be described in the Applications section of this paper. Our primary goal is to use the clustering on this data to extract information about biological processes which are happening during those time intervals. More specifically, we look at how expression rates change over time for a specific gene, then compare those changes to other genes which have a similar expression rate pattern. By doing so, we hope to find correlations between genes which have similar expression patterns, and infer what biological processes could cause this correlation.

We begin with a preliminary section detailing the background information needed in both biology and mathematics. Following the preliminaries is the main chapter describing clustering in its various forms, expounding on the ideas of two primary types of clustering, and also describing how one might know if the clustering used is sufficient or not. A discussion on how these theories are applied follows, explaining how we determined an optimal method for our data set and how this method was applied to extract protein function of different groups of genes.

Chapter 2

Preliminaries

2.1 Biological preliminaries

This work deals with gene expression data from the ciliate *Oxytricha trifallax*. Ciliates are unique organisms to study due to the presence of two types of nuclei instead of only one. They contain a diploid, zygotic germline nucleus known as the micronucleus (MIC) which is active during cellular conjugation, and they contain a somatic nucleus known as the macronucleus (MAC) which is formed from the MIC during conjugation and is the nucleus which is active during transcription [8]. During sexual reproduction, the MIC undergoes meiosis similar to other eukaryotic organisms, dividing into four haploid gametes, three of which degrade while the fourth splits mitotically to allow exchange with another cell [24]. After exchange, the haploid nuclei form together into a new MIC, which then splits mitotically as well to form two MIC's. After this split, the mother cell's MAC degrades, and one of the two MIC's is used to form a new MAC for the daughter cell. This is done by the process shown in Figure 1, where certain sequences referred to as internally eliminated sequences (IES) are spliced out, and the remaining macronuclear-destined sequences (MDS) undergo permutations and recombine together to form the new MAC. A full visual of this process is given in Figure 2.

The MAC of *O. trifallax* has a genome comprised of over 16,000 nano-chromosomes, and the genome of the MIC is far larger, with many base pairs getting removed during reproduction in the form of IES's [11]. The exact mechanism of rearrangement is yet unknown, and many researchers focus on studying it, such as Jonoska et al. [4] and Landweber et al. [8]. Another way this process can be studied is through looking at the expression rates of the genes in the MAC. As mentioned, the MAC is the active somatic nucleus, and by studying which MAC genes are actively expressing throughout different stages of sexual conjugation and rearrangement, we can better understand which biological processes are occurring at those times.

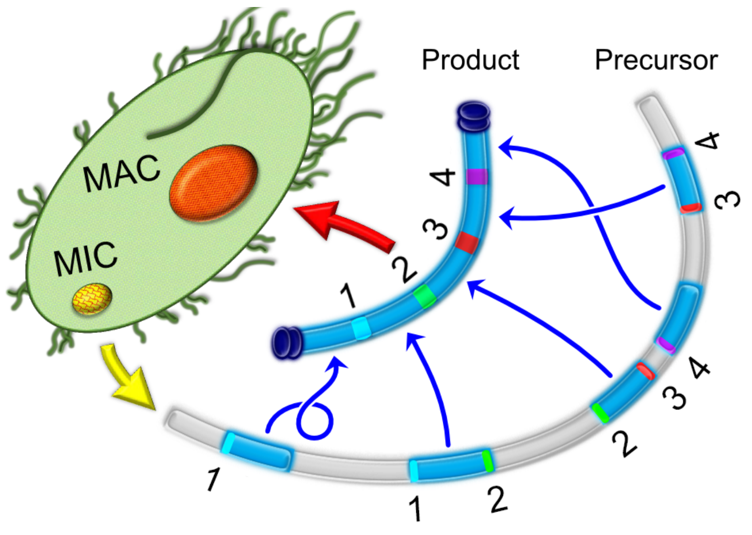

Figure 1. Cilitate rearrangement process. Genes going from the MIC to the MAC in sexual conjugation might be reordered and rearranged, along with having IES's between them.

2.2 Mathematical preliminaries

Clustering relies very heavily upon mathematical knowledge, particularly that of distance metrics, so first we must define the standard notion of a metric.

Definition 2.2.1. A *metric* on a set X is a function $d : X \times X \to \mathbb{R}$ which satisfies three conditions for all points $x, y, z \in X$:

$$\text{(i) } d(x,y) \geq 0, \text{ and } d(x,y) = 0 \text{ iff } x = y,$$

$$\text{(ii) } d(x,y) = d(y,x), \text{ and}$$

$$\text{(iii) } d(x,z) \leq d(x,y) + d(y,z).$$

The most well-known metric is the Euclidean metric d defined on $\mathbb{R}^n$ as

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$$

where x, y are vectors in $\mathbb{R}^n$. The Manhattan metric $d(x,y) = |x_1 - y_1| + \cdots + |x_n - y_n|$ is another commonly used metric. In statistics, the Pearson correlation coefficient is a very appreciated measure, and is defined as the covariance divided by the product of standard deviations. In formula form, the Pearson

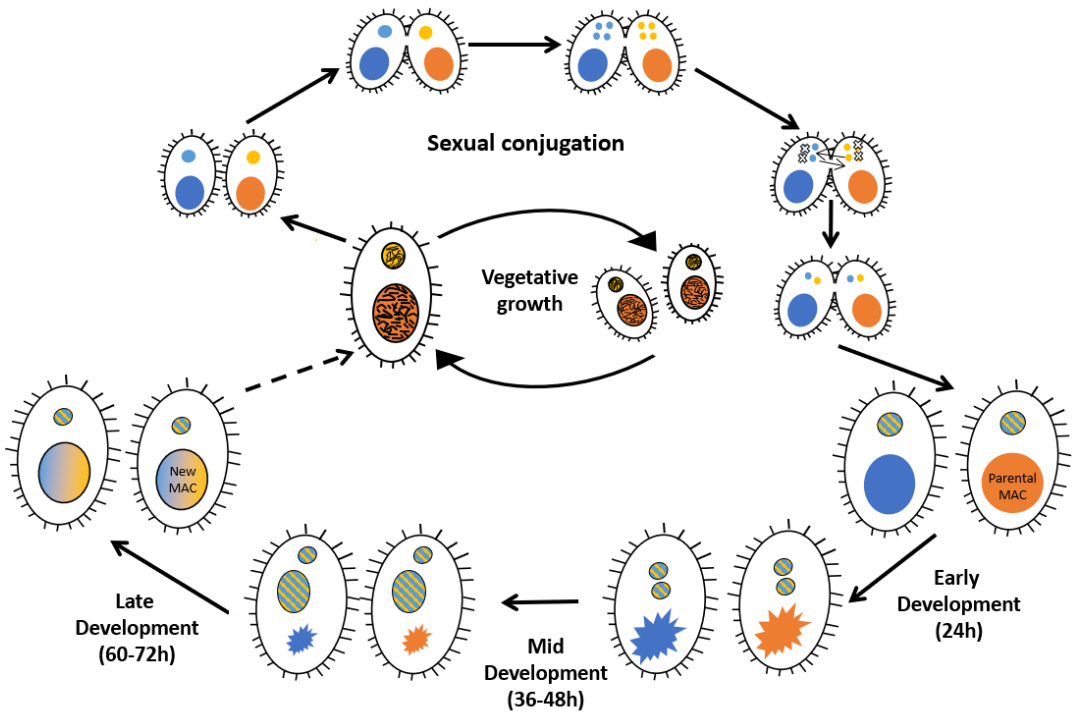

Figure 2. Life cycle of *Oxytricha trifallax*. The MIC undergoes typical meiosis, splitting into haploid micronuclei and undergoing exchange of those nuclei with another cell. The resulting new MIC splits mitotically to form a new MIC, via the process shown in Figure 1. Picture modified from [24].

correlation coefficient r_{xy} for two vectors $x = \langle x_1, \dots x_n \rangle$ and $y = \langle y_1, \dots, y_n \rangle$ is defined as

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{(\sum_{i=1}^{n}(x_i - \overline{x})^2)(\sum_{i=1}^{n}(y_i - \overline{y})^2)}}$$

where $\overline{x}$ and $\overline{y}$ are the mean of the entries of x and y respectively. One can quickly verify that if $x = y$, then $r_{xy} = 1$, which corresponds with perfect correlation between the two vectors. Naturally, a value of $r_{xy} = -1$ corresponds with the case when there is a negative correlation, and when $r_{xy} = 0$, there is no correlation between the two vectors. We note that since in a metric, $d(x, y) = 0$ if and only if $x = y$, we will say that the Pearson correlation gives rise to a metric, where the distance between two points x and y is $d(x, y) = 1 - r_{xy}$. Now $d(x, y) = 0$ when $x = y$, and d also obtains its largest value of 2 when $x = -y$.

One advantage of the metric defined by the Pearson correlation is its small range of $[0, 2]$, an advantage which will be covered later in Chapter 2. When using either the Euclidean or Manhattan metric however, two vectors may point in the same direction, but their magnitudes can be very different. The disadvantage of having very different magnitudes will be made clear in Chapter 5; however, for now we discuss how one might make all vectors to have the same magnitude. One might recall that any n-dimensional vector $v = \langle x_1, \ldots, x_n \rangle$ may be made to have a magnitude of exactly 1 by the use of vector normalization.

Definition 2.2.2. Let $v = \langle x_1, \ldots, x_n \rangle$ be a vector in $\mathbb{R}^n$. The *normalized* vector u of v is the vector

$$u = \frac{v}{|v|} = \frac{\langle x_1, \ldots, x_n \rangle}{\sqrt{x_1^2 + \cdots + x_n^2}}.$$

In short, normalizing the vector is just the process of dividing a vector by its length. By doing so, the direction of the vectors is preserved, but now the magnitude of each normalized vector will be exactly 1. Thus, when performing clustering as explained in Chapter 3, vectors would be clustered together strictly based on direction, rather than on both direction and magnitude.

Chapter 3

Clustering

The goal of clustering is to group together elements of a data set which are the most similar [3]. Intuitively, this can be as simple as grouping together individuals based on their answers on a personality test, or it can be used to group together children based on their performance in a class by assigning each student a letter grade. When looking at a group of vectors, or points in a metric space, the idea of clustering is fairly straightforward: simply group together the points which are closest to each other. However, as is pointed out in multiple papers ([3], [12]), not only are there many different algorithms for clustering, but none of them is universally better than the others. In fact, for a given problem one might find a clustering method which works, but for another problem that same clustering method might fail.

As for the usefulness of clustering, almost all fields which employ data analysis methods use one or more of the various clustering algorithms. Originally, clustering was introduced by anthropologists [10] and psychologists [25] in the 1930's as a means to measure correlations between different cultural groups and divide groups based on how like-minded they are, respectively. Over many years, clustering methods have evolved quite substantially from its origins, and now sees application in a plethora of fields. Among those applications, using it to find correlations between the expression rates of genes in a given genome is the main focus of this paper, and will be discussed in more detail in Chapter 4.

Two primary types of clustering are *partitional* and *hierarchical* clustering [3]. While there are actually quite a large number of clustering algorithms which have been presented over the years, these are two of the most common types, as well as the types most relevant to this project. In partitional clustering, one selects a number k to be the number of desired clusters, and separates the points into k partitions which will be the clusters. While the chosen number for k may be known, such as when a certain number of properties are being studies, it is often the case that k is unknown and must be derived from the data. In most examples of this type of clustering, each cluster $\mathcal{C}_i$ is assigned a center point c_i, and the points in the cluster $\mathcal{C}_i$ are closer to that given center than they are to any other cluster's center. In other words, each point x is in $\mathcal{C}_i$ if for our chosen metric d, $d(x, c_i) \leq d(x, c_j)$ for all $1 \leq j \leq k$. If there are two centers c_i and c_j to which x is closest and equidistant, then it is assigned to one of the two clusters at random. Note that while these centers do not necessarily need to be points from the initial set, they do need to lie within the domain of the given metric space.

For hierarchical clustering, two different systems may be used. The first, an *agglomerative* method, starts with every point of the data set being its own cluster. The two closest points are joined together as a single cluster, then the next two closest points are placed into a cluster, and the process repeats until the entire set of points is a single cluster. By selecting a value k for the number of desired clusters, one may freeze the analysis at the point with that many clusters and analyze what is in each set. The second system, a *divisive* method, starts with all points joined into a single cluster, then gradually splits groups of points until there are only singletons, or k clusters, remaining.

3.1 Hierarchical Clustering

The first type of clustering we discuss is hierarchical clustering, as discussed in Akman et al. [3]. For the sake of simplicity, we will only discuss the algorithm for an agglomerative hierarchical clustering, which begins with every point in a space as its own cluster, and gradually groups the points together based on their proximity to each other. Once all points have been joined into a single large cluster, the process ends. One advantage of this method is the ability to create an output each time two clusters join together, which is beneficial, since one might wish to check the output for multiple different numbers of clusters. The drawback to this method is that it is very computationally expensive, and running it a single time might take more computational power than running an alternative method multiple times.

To begin then, let $\mathbb{R}^n$ be our vector space, let $V = \{x_1, \ldots, x_m\}$ be a set of vectors on $\mathbb{R}^n$, let $\mathcal{C} = \{K_1, \ldots, K_n\}$ be our starting clusters on V. Since this is an agglomerative clustering, each cluster K_i is simply the singleton $\{x_i\}$ where x_i is a vector in V. We define a distance function d to be the Euclidean distance. This distance function is used to compute a *correlation matrix* M, where the entries a_{ij} of M are caluclated by taking the distance between the i'th and j'th points of V, so $a_{ij} = d(x_i, x_j)$. One can easily verify that the entries on the main diagonal are 0, since $a_{ii} = d(x_i, x_i) = 0$, and also that for every pair of points x_i and x_j, $a_{ij} = a_{ji}$. Thus M symmetric, and in practice it is calculated as an upper or lower triangular matrix to allow for a shorter calculation.

To cluster together multiple points, we now find a_{ij} where $i \neq j$ and $a_{ij} \leq a_{kl}$ for all $1 \leq k \leq n, 1 \leq l \leq$

n, and $k \neq l$; in other words, we find the smallest value in the matrix M that is not on the main diagonal. Then we form a new cluster $K_{ij} = \{x_i, x_j\}$, and remove K_i and K_j from $\mathcal{C}$. Furthermore, we must now define how to keep adding vectors to this cluster. Say we have formed the cluster K_{ij} as before, and let x_k be a vector whose correlation coefficient a_{ik} (or equivalently a_{jk}) satisfies that $i \neq k$ and $a_{ik} \leq a_{lm}$ for all $1 \leq l \leq n, 1 \leq m \leq n$, $l \neq m$, and x_l, x_m are not already in the same cluster. Then we remove K_{ij} and K_k from $\mathcal{C}$ and add the new cluster $K_{ijk} = \{x_i, x_j, x_k\}$. Intuitively, the way this method works is that we combine clusters K_i and K_j if there exists vectors $x \in K_i$ and $x' \in K_j$ such that $d(x, x')$ is the smallest distance between any two vectors which are not already clustered together.

Hierarchical clustering may be visualized by constructing a hierarchical tree [19], also called a *dendrogram*, as seen in Figure 3. To construct this tree, start with a set of disconnected vertices, each corresponding to a cluster. Whenever two clusters are joined together, we form a new vertex between the vertices of the corresponding vectors, and connect those vectors' vertices to the new vertex. Eventually, all points will be clustered together, resulting in a single vertex on top. One advantage of this visualization, and of this method in general, is that one doesn't need to perform multiple clusterings to change the number of clusters k. To observe the clustering using k clusters, one need only look at the level of the tree where k vertices are left.

3.2 Topological Data Analysis

Topological data analysis, or TDA, can be viewed as a form of hierarchical clustering. However, it is a more valuable tool, because it extracts more information about a data set than simply how points within the data set group together. TDA gives us the tools to better understand underlying structures of a data set, such as its shape, the connectivity of the clusters, and the holes or voids within connected components. The aim is to use a series of structures known as *simplicial complexes* which gradually become larger. This process is known as a *filtration*, which will be introduced following the notation of Nanda and Sazdanovic [18].

Definition 3.2.1. A *point cloud* P is a finite set of points $P = \{x_1, \ldots, x_k\}$, where each x_i lies in the Euclidean space $\mathbb{R}^n$.

Note that our point cloud is simply a realization of a given data set in Euclidean space. This point cloud can be thought of as a vertex set; however, to remain consistent we shall always refer to sets which would be a vertex set as a point cloud instead.

Definition 3.2.2. Let $\mathcal{C}$ be a collection of subsets of point cloud P. We say that $\mathcal{C}$ is a *simplicial complex* if

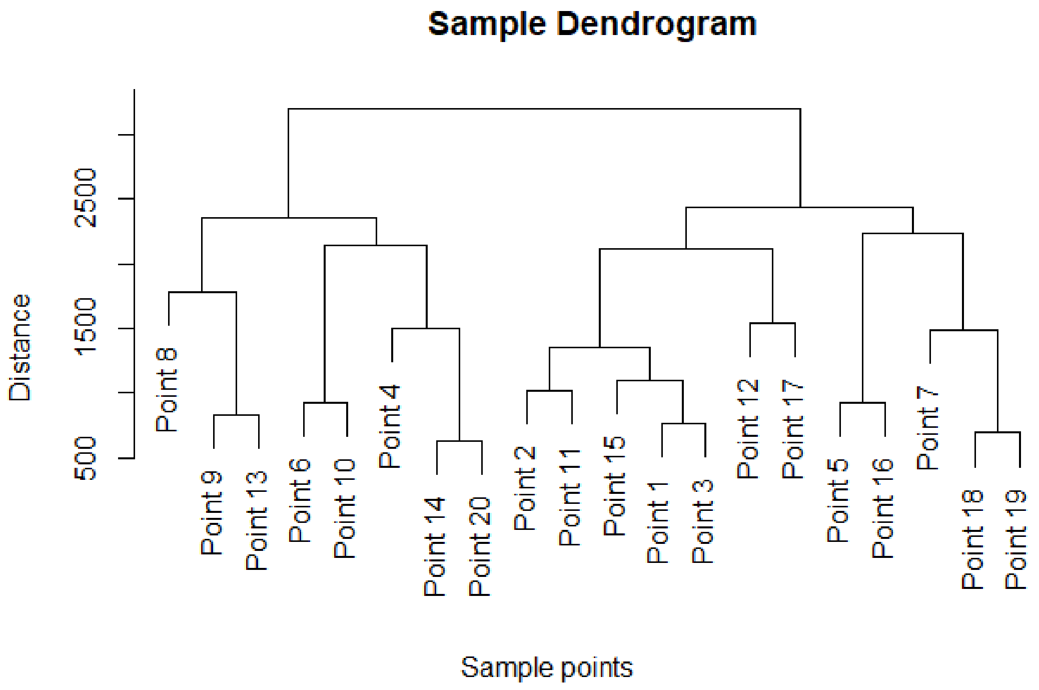

Figure 3. Dendrogram example on twenty randomly generated points. Points are clustered together based on the distance between them according to a defined metric. To find a clustering with n clusters, simply find the distance value at which there are n vertical lines.

every singleton of the point cloud P is an element in $\mathcal{C}$, and for every σ in $\mathcal{C}$, all subsets of σ also are in $\mathcal{C}$. Each subset $\sigma \in \mathcal{C}$ is called a *simplex*, and each subset of a simplex is called a *face* of σ.

Example 1. Let $P = \{a, b, c\}$ be a point cloud, and $\mathcal{C} = \{a, b, c, ab, bc\}$ be a collection of simplices. For the sake of brevity, braces around the subsets have been omitted, as each element of $\mathcal{C}$ is understood to be a set. It is clear that $\mathcal{C}$ is a simplicial complex since each one of the singletons a, b, and c is present, and for any simplex $\sigma \in \mathcal{C}$, all subsets of the simplex are also present in $\mathcal{C}$. However, the collections $\mathcal{C}_1 = \{a, b, ab, ac, bc, abc\}$ and $\mathcal{C}_2 = \{a, b, c, ab, bc, abc\}$ are not simplicial complexes. The first collection does not contain the singleton c, whereas the second one does not contain $ac \subset abc$, even though abc is in the collection.

If there are two collections of simplices, $\mathcal{C}_1$ and $\mathcal{C}_2$, such that $\mathcal{C}_1 \subset \mathcal{C}_2$, and both are simplicial complexes, then we say $\mathcal{C}_1$ is a *subcomplex* of $\mathcal{C}_2$, and we write $\mathcal{C}_1 \hookrightarrow \mathcal{C}_2$. As a relation, it is clear that $\hookrightarrow$ is reflexive, antisymmetric, and transitive, which means it is therefore a partial order. This partial order is what is now used to describe filtrations.

Definition 3.2.3. Let $n > 0$ be a natural number and $\mathcal{C}$ a simplicial complex. A *filtration* $\mathcal{F}$ is a collection of subcomplexes of $\mathcal{C}$, each one denoted $\mathcal{F}_i\mathcal{C}$, $i \leq n$ a natural number, and such that $\mathcal{F}_0\mathcal{C} = \emptyset$, $\mathcal{F}_n\mathcal{C} = \mathcal{C}$, and

$$\mathcal{F}_0\mathcal{C} \hookrightarrow \mathcal{F}_1\mathcal{C} \hookrightarrow \mathcal{F}_2\mathcal{C} \hookrightarrow \cdots \hookrightarrow \mathcal{F}_{n-1}\mathcal{C} \hookrightarrow \mathcal{F}_n\mathcal{C}$$

Since each $\mathcal{F}_i\mathcal{C}$ is a subcomplex of $\mathcal{F}_{i+1}\mathcal{C}$, and this filtration process is defined by a partial order relation, it should be clear that intuitively, the process of filtrations gradually builds larger simplicial complexes. This could be done trivially, by simply going from the empty set to the final simplicial complex $\mathcal{C}$, or it can be done by adding one simplex at a time to the simplicial complex.

Example 2. The filtration $\emptyset \hookrightarrow \mathcal{C}$ is the trivial filtration. For another example, let $P = \{a, b, c, d\}$ be the point cloud. Let $\mathcal{F}_1\mathcal{C} = \{a, b, c, d, ac\}$, $\mathcal{F}_2\mathcal{C} = \{a, b, c, d, ab, ac, bc, bd, abc\}$, and $\mathcal{F}_3\mathcal{C} = \mathcal{C}$ be the power set on P minus the empty set. Then $\emptyset = \mathcal{F}_0\mathcal{C} \hookrightarrow \mathcal{F}_1\mathcal{C} \hookrightarrow \mathcal{F}_2\mathcal{C} \hookrightarrow \mathcal{F}_3\mathcal{C} = \mathcal{C}$ is a filtration, as drawn in Figure 4.

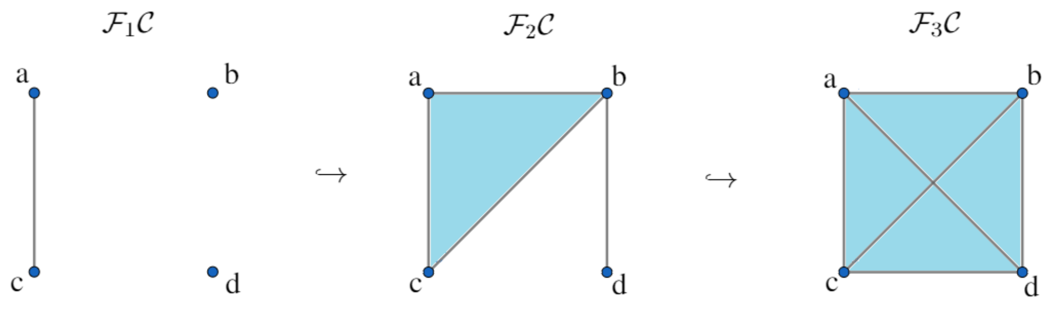
$\mathcal{F}_1\mathcal{C}$
a
b
c
d
$\hookrightarrow$
$\mathcal{F}_2\mathcal{C}$
a
b
c
d
$\hookrightarrow$
$\mathcal{F}_3\mathcal{C}$
a
b
c
d

Figure 4. Filtration using the point cloud $P = \{a, b, c, d\}$ as in Example 2.

The type of filtration we used for our research was a filtration known as the Vietoris-Rips filtration. While other filtrations exist, this was the only one employed in this project, and hence it is the only one we will discuss. For the Vietoris-Rips filtration, it is assumed that there is a point cloud P in $\mathbb{R}^n$, and that there is some metric defined on the point cloud. While any metric technically would suffice, it is often the case that we use the Euclidean metric. The first step of the filtration is to calculate all pairwise distances between points in the point cloud P. In other words, for all p_i, p_j in P, we calculate the distance $d(p_i, p_j)$ where d is defined by the chosen metric. To construct the filtration itself, let $\varepsilon_0 = 0$ be the initial epsilon value, and let some $\delta > 0$ be an incremental value. Define each subsequent ε_i as $\varepsilon_i = \varepsilon_{i-1} + \delta$, with $i > 0$, so that it is the value of the previous ε_i increased by the increment δ. At a given ε_i, we say that any group of points $\sigma = \{p_1, \ldots, p_n\}$, such that $\sigma \subseteq P$ is a simplex in the simplicial complex $\mathcal{F}_{\varepsilon_i}\mathcal{C}$ provided that for every pair of points p_j and p_k in σ, it is true that $d(p_j, p_k) < \varepsilon_i$.

A few observations may be made about the Vietoris-Rips filtration. First, if there are two values δ, ε such that $\delta < \varepsilon$, then any simplex in $\mathcal{F}_\delta\mathcal{C}$ is also in $\mathcal{F}_\varepsilon\mathcal{C}$. This follows by the fact that $d(p_1, p_2) < \delta < \varepsilon$ must follow for all points whose distance is smaller than δ. Also, for simplices K_1, K_2 such that $K_1 \subseteq K_2$, if K_2 is in some simplicial complex $\mathcal{F}_\varepsilon\mathcal{C}$, then so is K_1. Therefore the definition of the Vietoris-Rips filtration defines a true filtration of a simplicial complex.

Furthermore, these simplicial complexes can be used to infer various topological features about a data set. Let P be a point cloud with n points, and let $G = (P, E)$ be a graph on this point cloud, where the edge set E begins as the empty set. At the beginning of the filtration, we see that $\varepsilon_0 = 0$ and therefore no points are connected together, hence there exist n distinct components of the graph. We say that this results in the *birth* of n components of the *0-dimensional persistent homology.* This 0-dimensional persistent homology allows us to the count the number of connected components of our graph. As the ε value increases, components begin to connect together as the distance between them becomes less than ε, and edges are formed between them and added to the edge set E. When there are k connected components remaining, and two points p_1 and p_2 from separate components, $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, are connected to form the edge $e = \{p_1, p_2\}$, then the component $\mathcal{C}_3$ will be produced, containing both $\mathcal{C}_1$ and $\mathcal{C}_2$. There would then be only $k - 1$ connected components in the graph, corresponding in the *death* of one of the components $\mathcal{C}_1$ or $\mathcal{C}_2$ of the 0-dimensional persistent homology; say it resulted in the death of C_1, then the birth of $\mathcal{C}_3$ is inherited as the birth of the other component $\mathcal{C}_2$. For a component of the persistent homology, we denote with ε_b the value at which the component was birthed, and with ε_d the value at which the component died. Then we call the interval $[\varepsilon_b, \varepsilon_d]$ the *lifespan* of the component.

Furthermore, we may also observe generators of the *1-dimensional persistent homology*, which are cycles within the graph. When a cycle is formed, it results in the birth of a generator of this persistent homology, and when all triangles within the cycle are formed, this results in the death of the generator. An example of cycles is given in Figure 5, where one may observe both a triangle and a much larger cycle being formed at the value ε_3. The cycle corresponding to the triangle is immediately filled in and dies, whereas the larger cycle will die at a much later ε value.

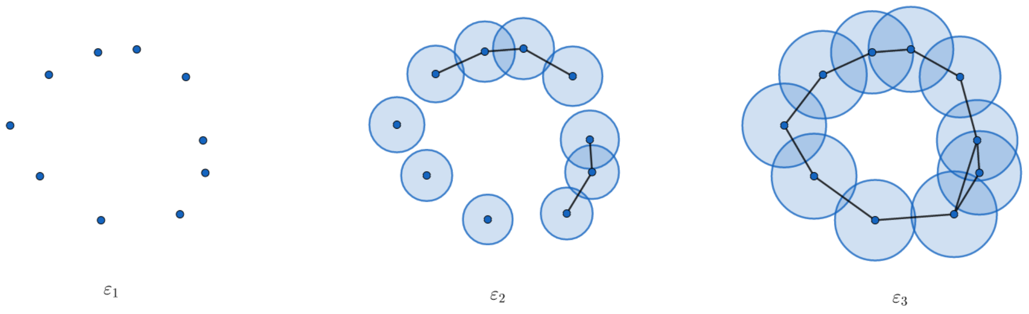

Figure 5. Visual example of persistent homology. At ε_1, each point represents the birth of an element of the 0-dimensional homology. At ε_2 several elements become part of the same connected component, resulting in the death of multiple components of the simplicial complex. However, no generators of the 1-dimensional homology have formed yet. At ε_3, all points are part of the same connected component, which means there is only one component representing a generating class of the 0-dimensional homology. Also, a generator of the 1-dimensional homology has now formed as a large cycle, which will persist until larger ε value. There was also a triangle which was birthed at ε_3, but immediately died.

Many papers have found ways of visualizing the births and deaths of the classes of persistent homologies over time. One very useful means by which this is done is through the use of barcodes, as described in a paper by Ghrist et al. [15]. This method creates a visual similar to a bar graph, with the x-axis being the ϵ value in a Vietoris-Rips filtration, and the y-axis being the elements in the homology which are birthed. For each element in the homology, a bar begins at $\varepsilon = \varepsilon_b$, the value where that element is birthed, and continues until $\varepsilon = \varepsilon_d$, the value at which its death occurs. Typically, a separate set of bars is used for each dimension of persistent homology.

Another way of visualizing a persistent homology is through persistence diagrams [18]. In this visualization, we look at the Cartesian plane, where the x-axis corresponds to the time of birth, the y-axis corresponds to the time of death, and we draw the line $y = x$. For each element $\mathcal{C}$ in a given homology, either the 0-dimensional homology or the 1-dimensional homology, we look at the lifetime of $\mathcal{C}$, $[\varepsilon_b, \varepsilon_d]$, and we plot a point in the persistence diagram at the point $(\varepsilon_b, \varepsilon_d)$. An example of such a plot is given in Figure 7. For the 0-dimensional homology, all components are birthed at $\varepsilon_b = 0$, so all points in the persistence diagram for the 0-dimensional homology will be of the form $(0, \varepsilon_d)$. Hence all points in the persistence diagram for the 0-dimensional homology will lie on the y-axis. We also note that it can never be the case that $\varepsilon_b > \varepsilon_d$,

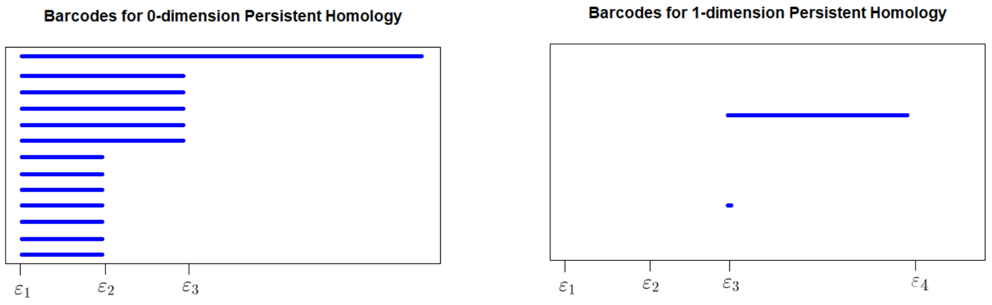

Figure 6. Barcode graph. Roughly matches the sample point cloud in Figure 5. On the left is a graph measuring the lifetime of components of the 0-dimension persistent homology. The lifetimes of these components start at the initial value $\varepsilon = \varepsilon_1$ and persist until the ε value where they are combined with another component. The top bar exists because one the final connected component will exist at every ε past ε_3. On the right are the barcodes measuring the lifetimes of elements of the 1-dimensional persistent homology. The dot represents the triangle from Figure 5, since it dies immediately after it is birthed. The bar on top represents the much larger cycle, as it will be filled at a much later value $\varepsilon = \varepsilon_4$.

since a component must be birthed before it can die, hence all points in a persistence diagram must lie on or above the line $y = x$.

3.3 K-means Clustering

K-means clustering has been a consistently prominent clustering tool ever since it was introduced. Though the algorithm was introduced in 1957 by Lloyd [16], the first known publication of the algorithm wasn't until 1965 by Forgy [14]. The basic premise of this method relies on dividing the metric space $\mathcal{M}$ into k regions which are called clusters, with each cluster being defined by its center. To begin this method, first one defines a metric space $\mathcal{M}$ with a metric d based on the point cloud P generated by a set of data. The most obvious choice for this metric would be the standard Euclidean distance when $P \subseteq \mathbb{R}^n$, but other metrics work just as well depending on the need. After choosing a metric d on the metric space M which contains P, k random points are selected in the space to be the centers, and the set of centers is denoted $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$. One important note is that these centers do not have to be one of the points in the original point cloud; so long as they are points within the metric space, they are acceptable as centers. However, one may choose to select centers within the smallest polytope $P' \subseteq M$ which contains P. Then for each point x in the point cloud

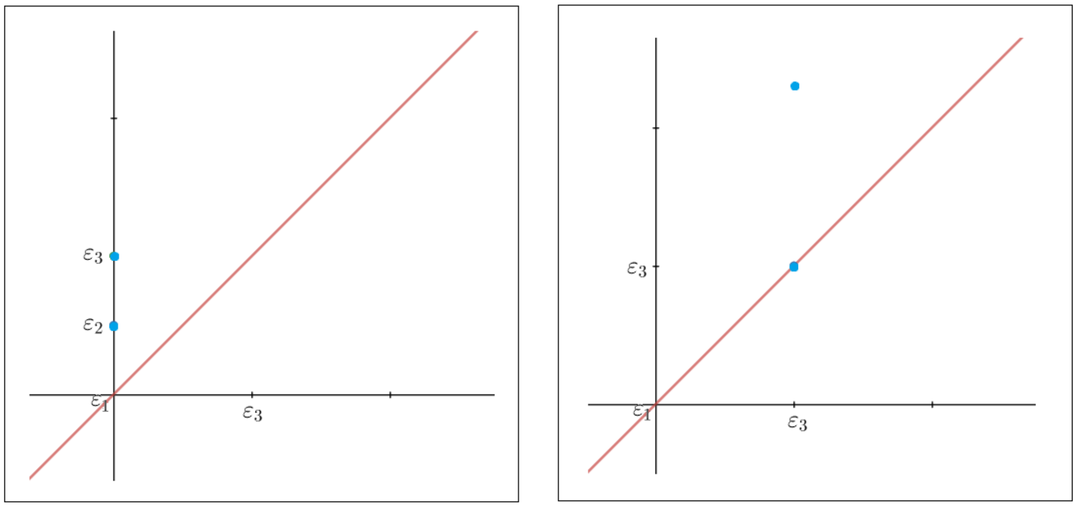

Figure 7. An example of a persistence diagram. The x-axis represents the time of birth, and the y-axis represents the time of death. The line $y = x$ is given as a reference, since all components of a persistent homology should always die either at the same ε at which they are born, or at a later ε. Hence all points should be either on or above the line $y = x$. On the left is a persistence diagram measuring the lifetime of the components of the 0-dimension homology from Figure 5. All components are birthed at $\varepsilon_1 = 0$, so these points all lie on the y-axis. On the right is the diagram measuring the lifetime of components in the 1-dimensiona homology from Figure 4. One dot that lies on the line $y = x$ both was born and died at ε_3, corresponding to the triangle in Figure 5. The much larger in cycle Figure 5 was born at ε_3, but persists until a much later $\varepsilon = \varepsilon_4$, hence it is placed at the point $(\varepsilon_3, \varepsilon_4)$.

P, the distance from that point to each center c_i is calculated using the defined metric, $d(x, c_i)$. Each cluster K_i with center c_i is then defined as

$$K_i = \{x : d(x, c_i) < d(x, c_j), \text{ for all } 1 \leq j \leq k, i \neq j\}.$$

After each point has been assigned to a cluster, a recalculation is performed. Since the centers were randomly selected, they will likely not correspond with the best possible clustering. To account for this, the centers are discarded, and a new set of centers is calculated based on the current clusters. To each cluster $K_i = \{x_1, x_2, \ldots, x_n\}$, the center is calculated by the formula

$$c_i = \frac{1}{n}\sum_{j=1}^{n} x_j$$

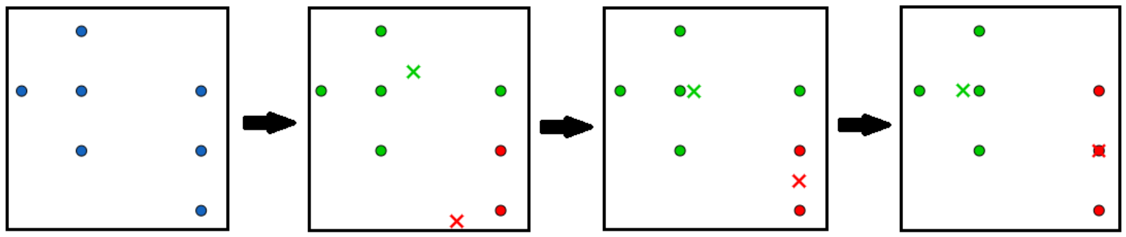

Figure 8. An example of k-means clustering. As can be seen, the initial centers are random, and points are assigned to whichever center they are closest to. This may not give an accurate clustering, so centers are recalculated using the points in the resulting clusters. The process of recalculating centers repeats until no further changes are made to the centers.

and all points are then reassigned to new clusters according to the new centers using the same method as before. This process is repeated until the centers do not change during recalculation, or until a set number of iterations is performed.

Note that because the initial points are chosen at random within the domain, two different runs of the algorithm could potentially conclude with two different clusterings. Because of this, it is typically useful to run the k-means algorithm multiple times, to ensure that the final clustering is as close to ideal as possible. Unfortunately, it is not possible to guarantee that an ideal clustering is obtained, so this fact must be taken into consideration when using the k-means algorithm.

3.4 Statistical Tests

After choosing a method of clustering, one must ask further: how do we know these clusters are any good? In other words, how do we know these clusters could not be formed by simply picking random points out of the given data set. While one might view this as highly unlikely, it still begs to be tested for proper verification. For this, the method of Gene Set Enrichment Analysis (GSEA) is introduced, as described in Subramian et al. [21]. In the biology world, one would typically use GSEA to test for similarities between two data sets; in particular it compares a new, unknown data set against a pre-existing, well-studied one. However, we introduce our own variation of it which will instead compare our clusters to the entire data set in order to validate the clusters.

To begin explaining our variation of GSEA, first let P be a point cloud with n points, with each point p_i

containing m entries. The point cloud P corresponds to a matrix D which has n rows and m columns. Let C be a subset of P with $k < n$ points; so $\mathcal{C}$ can be taken to be our cluster containing k points from P, and we wish to test whether or not this subset $C \subseteq P$ could have been chosen randomly. Note that C can also correspond to a matrix D' with k rows and m columns, and that $D \backslash D'$ is also a matrix with $n - k$ rows and m columns. We define an ordering on D as follows. For the i'th column vector c_i in C, determine the average of all the entries in that vector, and denote it $\overline{c_i}$; so $\overline{c_i} = \frac{c_{i_1} + \cdots + c_{i_k}}{k}$, where each c_{i_j} is the j'th entry of the i'th column vector. Let $\mu_C = \{\overline{c_1}, \ldots, \overline{c_k}\}$ be the set of all such averages, and let $\mu = \frac{1}{m}\Sigma_{i=1}^{m}\overline{c_i}$ be the average of these averages. This value μ is calculated because for every column c_i, if $\overline{c_i} < \mu$, then the entries of c_i contain values which are above average, relative to the other column vectors. This allows us to partition our column vectors into vectors which have a above-average values, and those which have below-average values. We partition the indices of the column vectors of C into two sets: $C_1 = \{i | 1 \leq i \leq m, \overline{c_i} \leq \mu\}$ are the indices of those columns which have below-average values, and $C_2 = \{i | 1 \leq i \leq m, \overline{c_i} > \mu\}$ are the indices of those columns which have above-average values.

Using this partition, we may now assign a value to each row vector v_j in D. Note that this assigns a value to each vector in the data set, not just the cluster. So to each vector v_j in D, we define the value

$$a_j = \frac{\sum_{i \in C_2} D_{ij}}{\sum_{i \in C_1} D_{ij}}$$

where $1 \leq j \leq n$. For any two row vectors v_i and v_j in V, we say that $v_i < v_j$ if $a_i < a_j$. If it is the case that $a_i = a_j$, then one may choose randomly whether $v_i < v_j$ or $v_j < v_i$. By rearranging D according to this ordering, it should be the case that all vectors in C are moved to the beginning of the ordered set $\overline{D}$, assuming it is ordered from the largest value to the smallest value. If instead we order from smallest value to largest value, then all vectors in C are moved to the end of the ordered set $\overline{D}$. To verify that the data points from the cluster $\mathcal{C}$ are at the end of the ordered set $\overline{D}$, we introduce the recursive GSEA function $G : [0, n] \to \mathbb{R}$, where $[0, n]$ is an integer interval. Set $G(0) = 0$, and say

$$G(i) = \begin{cases} G(i-1) + a & \text{if } v_i \in \mathcal{C} \\ G(i-1) - b & \text{if } v_i \notin \mathcal{C} \end{cases}$$

where each v_i is a vector in $\overline{D}$, $a = \sqrt{\frac{n-k}{k}}$ and $b = \sqrt{\frac{k}{n-k}}$. A quick combinatorial verification would show that this function starts and ends at 0, provided it goes through every interval $i \in [1, n]$. Because this function is defined on a finite set, it will clearly achieve both a maximum and minimum value. Once we

	c_1	c_2
v_1	7	2
v_2	1	5
v_3	3	5
v_4	4	1

$\rightarrow$

	c_1	c_2
v_4	4	1
v_1	7	2

	c_1	c_2
v_3	3	5
v_2	1	5

$\rightarrow$

	c_1	c_2	a_i
v_4	4	1	4
v_1	7	2	3.5
v_3	3	5	0.6
v_2	1	5	0.2

Figure 9. Example of ordering. The data on the left, consisting of four vectors and two columns, are clustered into two obvious clusters: one cluster with high values in c_1, and one cluster with high values in c_2. The table on the right is ordered largest to smallest according to the ratio a_i.

know the largest deviation G takes from 0, whether it be the maximum or minimum, we say that $\max\{|G|\}$ is the *enrichment score* (ES).

The ES is the primary value which we will use to determine how useful our clusters are. To do this, we generate p random clusters using vectors from D, all of size k. In this way we may keep the calculations consistent with our non-random cluster which we wish to evaluate. For each of these p clusters, we use the same method outlined above to generate its ES. Once all p scores have been obtained, the ES of our original cluster is then compared to the rest; if it is greater than a certain percentage of the random clusters, then the cluster is deemed good; otherwise, it is discarded.

To better explain why this method works, it would be useful to visualize the ES graph over time. As one might see in Figures 10 and 11, one can see how an ES graph would look using a properly ordered set, and a randomly ordered one (or one ordered based on random clusters). In Figure 10, the same ordering from Figure 9 is used, as well as a random ordering which is not shown. The comparison shows that the ES of the proper order is higher, which is expected. For an actual plot using real data, we present Figure 11. In this image, the left graph demonstrates an actual cluster's ES, while the right graph demonstrates a random cluster's ES. Because the ordering introduced theoretically should move all the vectors in the cluster to the top or bottom of the ordered data set (in this case, the bottom), the gradual decrease followed by a sudden increase is expected: the recursive function G keeps decreasing until the end, where it suddenly increases extremely fast. However, this is not the case in a random cluster. When one tries to reorder the data set based on a random cluster, it cannot reliably place all the vectors in one location, since there is no consistent pattern in the entries of its vectors. Hence, the random cluster's vectors are scattered throughout the data set, even after reordering, causing the graph seen on the right, and consequentially causing the low ES.

	c_1	c_2
v_1	7	2
v_2	1	5
v_3	3	5
v_4	4	1

$\rightarrow$

	c_1	c_2	a_i
v_4	4	1	4
v_1	7	2	3.5
v_3	3	5	0.6
v_2	1	5	0.2

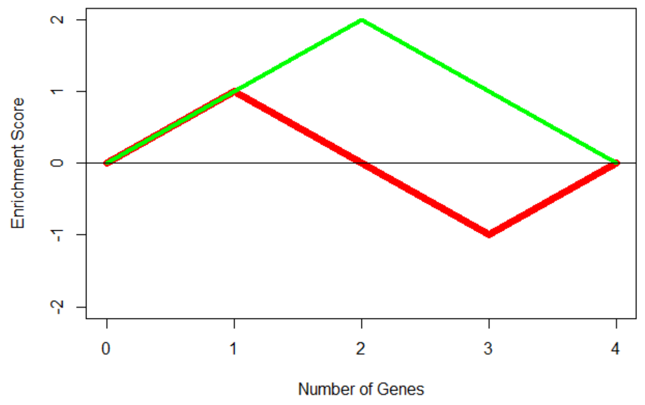

Figure 10. Example of enrichment score graph. The ordering is the same as in Figure 9. The graph gives the ES track of this ordering (green) and the ES track of a random ordering (red).

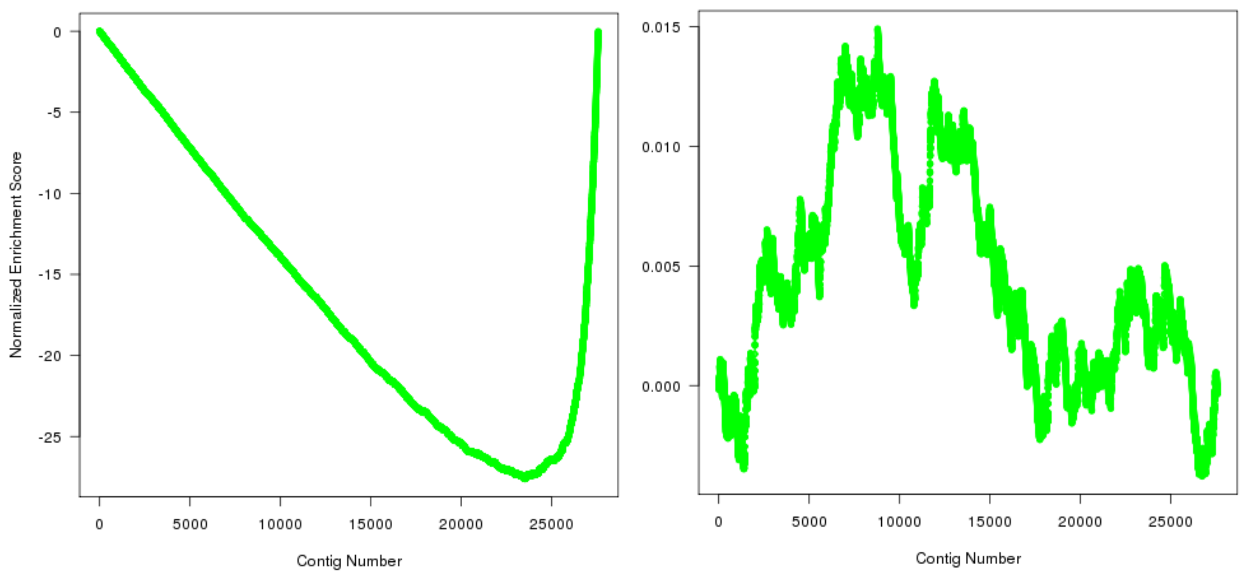

Figure 11. Visualization of the enrichment score function. Since all genes in our cluster should be at the beginning or end of an ordered list, the graph should either increase rapidly then decrease back to zero slowly, or decrease slowly at the beginning and increase rapidly at the end. The left image represents the latter case, where the cluster's genes were found at the end of the ordered list. For comparison, a random cluster is represented on the right, where genes are scattered throughout the cluster since there is no easily-defined ordering.

Chapter 4

Clustering analysis of the gene expression rates of *O. trifallax* during development

As mentioned in Chapter 1, we used clustering to analyze a set of gene expression data from the ciliate *Oxytricha trifallax*. The data set contains expression rates from approximately 27,000 genes, all of which measured at six different times points across 72 hours during cellular conjugation. The data was normalized using the DeSEQ 2 package in R [17] to account for abnormally high or low expression rates. Furthermore, three trials were done to measure the expression rates, and we considered the mean, variance, and index of dispersion (variance divided by mean) of these three trials. The primary goal of this analysis was to use clustering to group together those genes which expressed highly at the same time points in order that we might find correlations between those genes. In particular, we wished to determine which biological processes are happening in the cell at specific times. We used a method known as *gene ontology*, which will be discussed in Section 4.2, to use correlations between these genes to obtain a list of possible protein functions. By generating such a list, we could obtain information about cellular processes going on in the cell during conjugation.

4.1 Determining an Optimal Clustering

Before we analyzed any clustering, we had to first determine which clustering algorithm best fits our data set. While technically many different algorithms could have worked, we ideally wanted an algorithm which was both fast and gave statistically significant clusters. Furthermore, it was also necessary to determine what number of clusters we wanted, and why we wanted this number. Two conditions were considered for this ideal number: first, for each of our six time points, we wanted a cluster whose genes express highly at that time point; second, we did not want multiple clusters which were too similar to each other.

Initially we tried using hierarchical clustering on the data set to cluster together the genes, primarily because we were unsure of how many clusters we needed. As pointed out earlier, one advantage of hierarchical clustering is that the analysis is only run one time, and it is easy to adjust the number of clusters afterwards by simply looking at the hierarchical tree. We used a self-written script in Python utilizing familiar packages such as SciPy [23] and NumPy [20] to do this analysis. However, this analysis proved to be less than useful. When clustered, most of the genes in our data set were all added to a single large cluster, with many singletons and very small clusters left behind. While this does imply the existence of outlying genes with irregular expression rates, it does not tell us much about the data as a whole.

Alongside hierarchical clustering, we also did analysis involving TDA to generate the clusters. However, with TDA we were primarily hoping to extract information about the structure of the data, rather than actually cluster together genes of similar expression rates. In doing so, we utilized two programs primarily: Javaplex [22] and Ripser [6]. However, due to the size of our data set, we could not find a program that could run TDA on the entire set. Instead, we ran each program multiple times on random subsets of the data, ranging from 200 points to 4,000 points. Samples from these analyses can be viewed in Figures 12 and 13. While some interesting structures might be appearing, it does not appear than any major anomalies were found in the data. This fact, combined with the inability to feasibly run the programs on the full data set, led us to move away from further analysis using TDA.

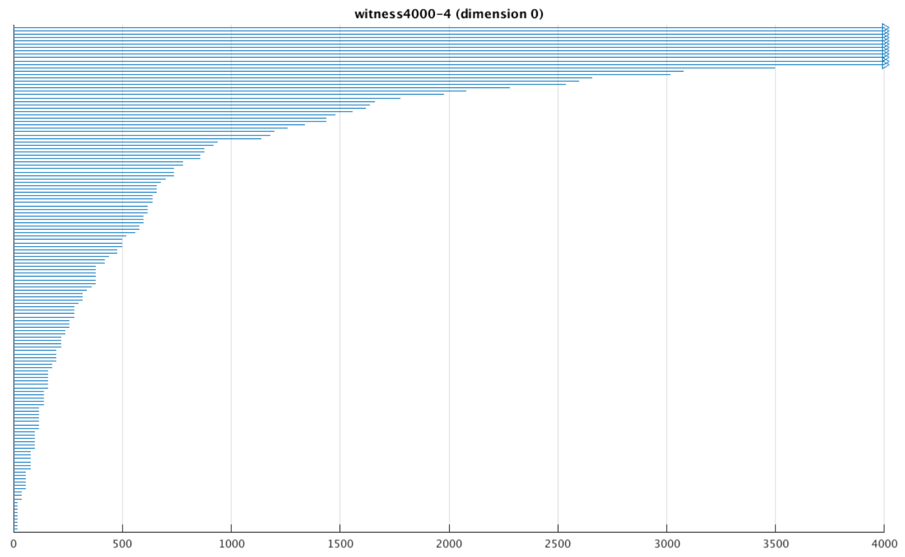

Figure 12. Barcode diagram on a 4000 point sample. These 4000 points were taken from our data set, and multiple such subsets were considered. Generated using Javaplex [22].

Finally, we instead decided to use k-means clustering and see if we could produce more optimal results.

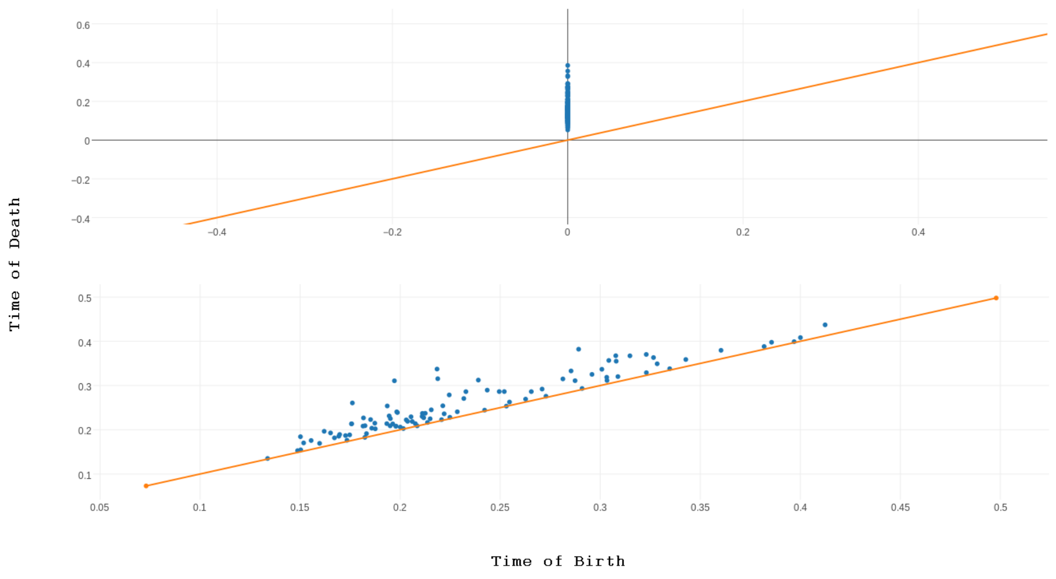

Figure 13. Persistence diagram using 200 random points from our data set. The top diagram represents the 0-dimensional persistent homology, while the bottom diagram represents the 1-dimensional persistent homology. Generated using Ripser [6].

This clustering was done using Multi-experiment Viewer (MeV) [2]. Initially, the results ended the same: a few small clusters, and one large cluster. However, after applying vector normalization to our data as described in the preliminaries, this clustering provided very nice clusters. Figure 14 shows these clusters in a two-dimensional graph, where the x-axis corresponds with time, and the y-axis corresponds with the expression rate. The red line in the middle of each cluster gives the average expression value over time. As one can see, there are multiple distinct shapes appearing in this clustering; in particular, for each of the six measured time points, there is at least one graph for which that time point's average expression value is very high compared to the expression value at other time points. The data was also observed at varying numbers of clusters, from a grouping as small as four clusters (see Appendix A, Figure A.02), to some as large as thirty-five clusters (see Appendix A, Figure A.01). Ultimately, it was determined that since eight clusters gave us all desired shapes, in particular the shapes that have exactly one spike point, this was the most useful number of clusters.

To follow up with k-means, what was needed was to determine whether these clusters were statistically good or not. When applying GSEA as described in Section 2.3, the results in Figure 15 occurred, and it

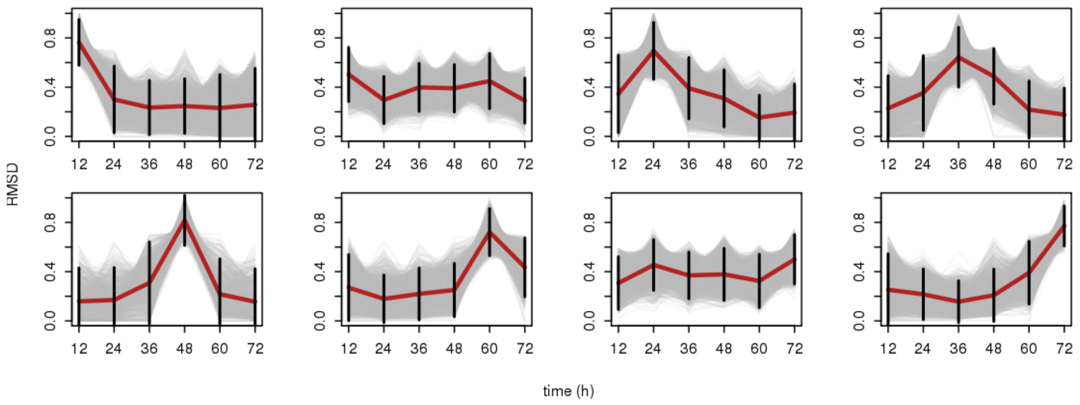

Figure 14. Optimal clustering of eight clusters. When determining an optimal number of clusters, it is ideal to find a variety of shapes with no repeats. Eight clusters gives this, so it is deemed an optimal number. Generated using MeV [2].

was easy to conclude that our clusters could not have been generated randomly. Therfore, it was determined that k-means provided a good clustering that would provide useful results. Even though it was determined that eight clusters was optimal, GSEA was also tested on all other cluster sizes. The same results occurred across all numbers of clusters, and it was concluded that k-means would give a good clustering regardless of the number of clusters.

4.2 Gene Ontology

As was discribed in the introduction, the ultimate goal of the project is to extract protein function data given a set of gene expression data. While determining the exact protein function of a given gene is a much more difficult problem to solve, it is possible to determine how likely it is that a given set of genes codes for a set of proteins. In mathematical terms, let $G = \{g_1, \ldots, g_n\}$ be a set of genes, $P = \{p_1, \ldots, p_m\}$ a set of protein functions, and $f : G \to P$ a mapping corresponding with a gene from G coding for a protein function in P. We can then determine for each p_j in P how likely it is that at least one g_i in G is such that $f(g_i) = p_j$.

To acheive the above goal, we introduce a method known as *gene ontology*. Gene ontology was introduced by many scientists in [5], and the basic goal they describe is to connect, or unify, all living organisms together by correlating their genomes. In short, gene ontology, or GO, compares our group of genes to a more well-

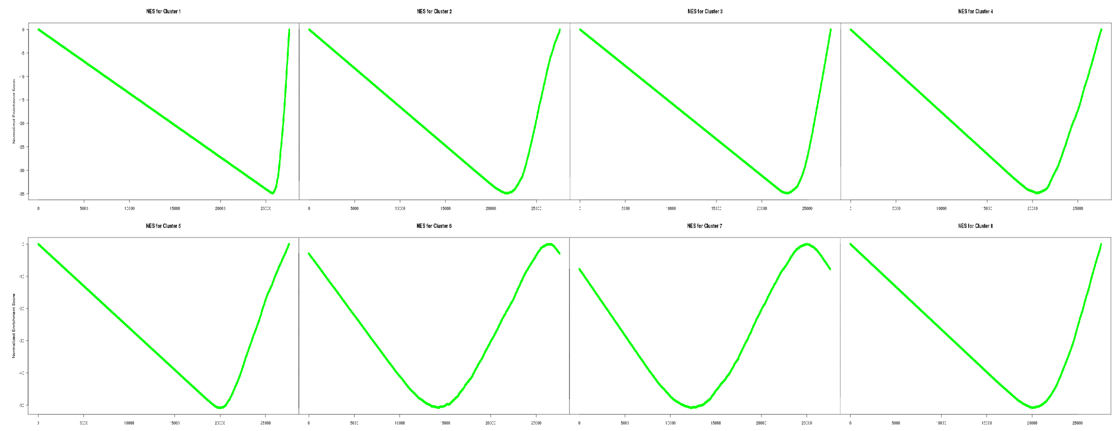

Figure 15. ES results from the clusters in Figure 14. Observing these graphs, one can see that all the genes in the data set were sorted fairly close together. This implies that our clustering is good, since the genes within any given cluster are not scattered throughout a sorted list.

studied genome, such as human or yeast. Since it is well-known which genes code for what protein functions in these organisms, if a correspondence can be made between our clusters of genes in *Oxytricha trifallax* and a group of genes in one of these well-studied organisms, we can determine how likely it is that at least one gene in our cluster codes for a specific protein function.

The first step of gene ontology is to use the program known as the Basic Local Alignment Search Tool, or BLAST [1], to match a gene sequence from our organism against all gene sequences of humans or yeast. By using this comparison and flagging all human and yeast sequences which match above a given minimum percentage of base pairs with the *O. trifallax* sequence, it can be determined which human and yeast sequences are most like the given *O. trifallax* sequence. Of important note is that this is not a bijective mapping, as each *O. trifallax* sequence may match with multiple human or yeast sequences. After a set of human sequences and yeast sequences have been determined, one may then use the Gene Ontology Resource database [9] to obtain a list of GO terms to which those sequences correspond. A given sequence may correspond to one or many different GO terms, so there is not a bijective mapping here either. Once a list of GO terms has been determined, each GO term corresponds to a set of protein functions, each of which is then assigned to the original gene from *O. trifallax*. By comparing against both human and yeast genomes, the possible protein functions can also be narrowed down to only those which were mapped to in both cases.

After all genes from *O. trifallax* have been given a list of possible protein functions, each cluster is then given a long list of possible protein functions based on which functions were mapped to by the genes within the cluster. Each function is also given a p-value weight based on how many genes within the cluster mapped to it; for example, a function which was mapped to by one hundred genes will have a lower p-value than a function mapped to by only one gene. Using these p-value weights, the most likely protein functions can be determined for the cluster as a whole. This yields biological insight into what the cell is doing at a given time, since the clusters are determined by expression over time. There were two outputs for results: one, a table listing all protein functions and their p-values, and another showing a word cloud [13] of the most likely protein functions of the cluster. Figure 16 gives an example of a word cloud, while Figure A.03 in Appendix A shows the word clouds for all eight clusters which were generated in Figure 14.

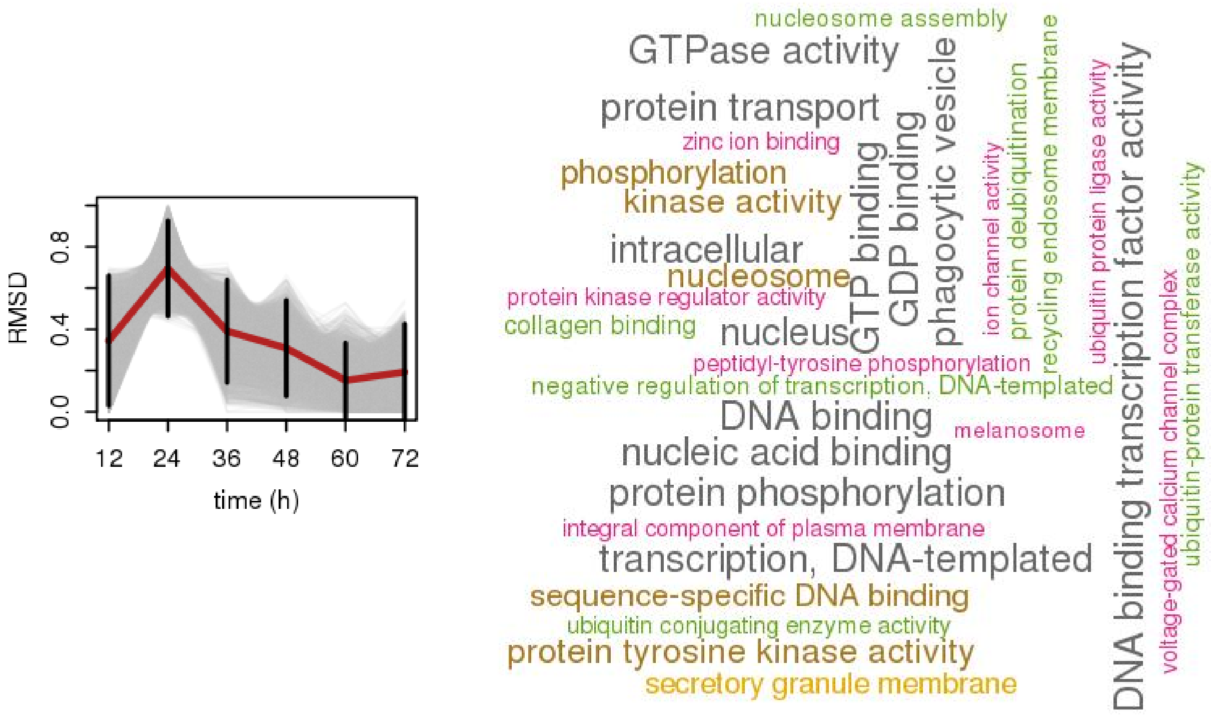

Figure 16. Example of a word cloud. To each cluster, a word cloud is associated showing which protein functions are most likely to be coded for by the genes of the cluster. The larger the word of the protein function, the more likely it is to be coded for.

4.3 Nested genes

One additional application which was looked at was the concept of nested genes as described in Braun et al. [7]. In ciliates, when a macronucleus is formed from a micronucleus, the genes can be rearranged as seen in Figure 17. However, certain genes found in the MAC are nested entirely within other genes in the MIC, and in certain cases a gene may have multiple genes sequentially nested within it. Given a gene g, the number of recursively embedded genes within g give an *insertion depth index*, or IDI [7], of g. For example, if g_1 is embedded in g, and g_2 is embedded in g_1, then g has IDI of 2; however, if both g_1 and g_2 are embedded in g, but neither is embedded in the other, then g only has an IDI of 1.

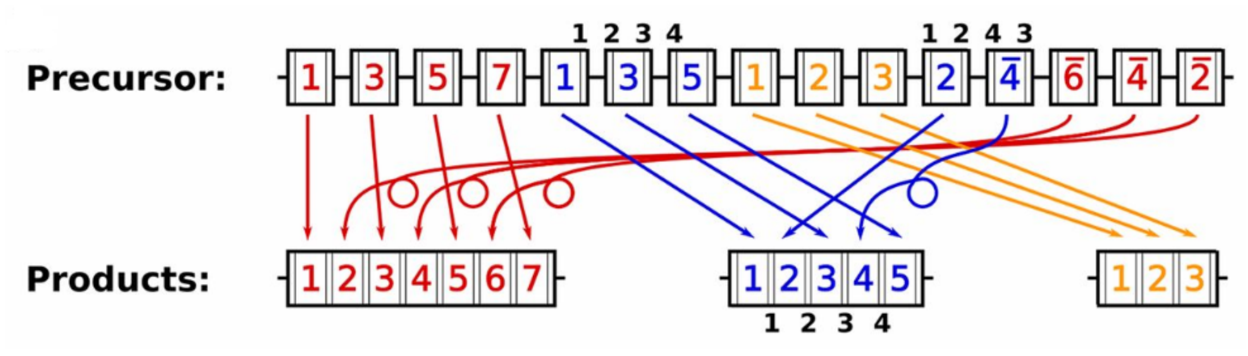

Figure 17. Example of nested genes. The red sequence has an IDI of 2, since there are two layers of genes (blue and orange) nested within it in the precursor, while the blue sequence has one set of genes nested within, hence it has an IDI of 1. Image from Braun et al. [7]

A question which arises with IDI genes is whether or not there is a correlation between the outer gene and those genes nested within it. In *O. trifallax*, there are two genes which have an IDI of 4, meaning that there are four sequential embeddings within them, along with some other non-sequentially embedded genes. In Figure 18, a tree is shown comparing the 2-dimensional expression graph of each IDI 4 gene with all of its nested genes. While one could hypothesize that such nested genes could follow a pattern, looking at Figure 18, it would seem there is virtually no such correlation. Looking further into the word clouds could have some information as well; however, due to the variety of shapes, and therefore different clusters, it would be unlikely that such an analysis would provide useful insight.

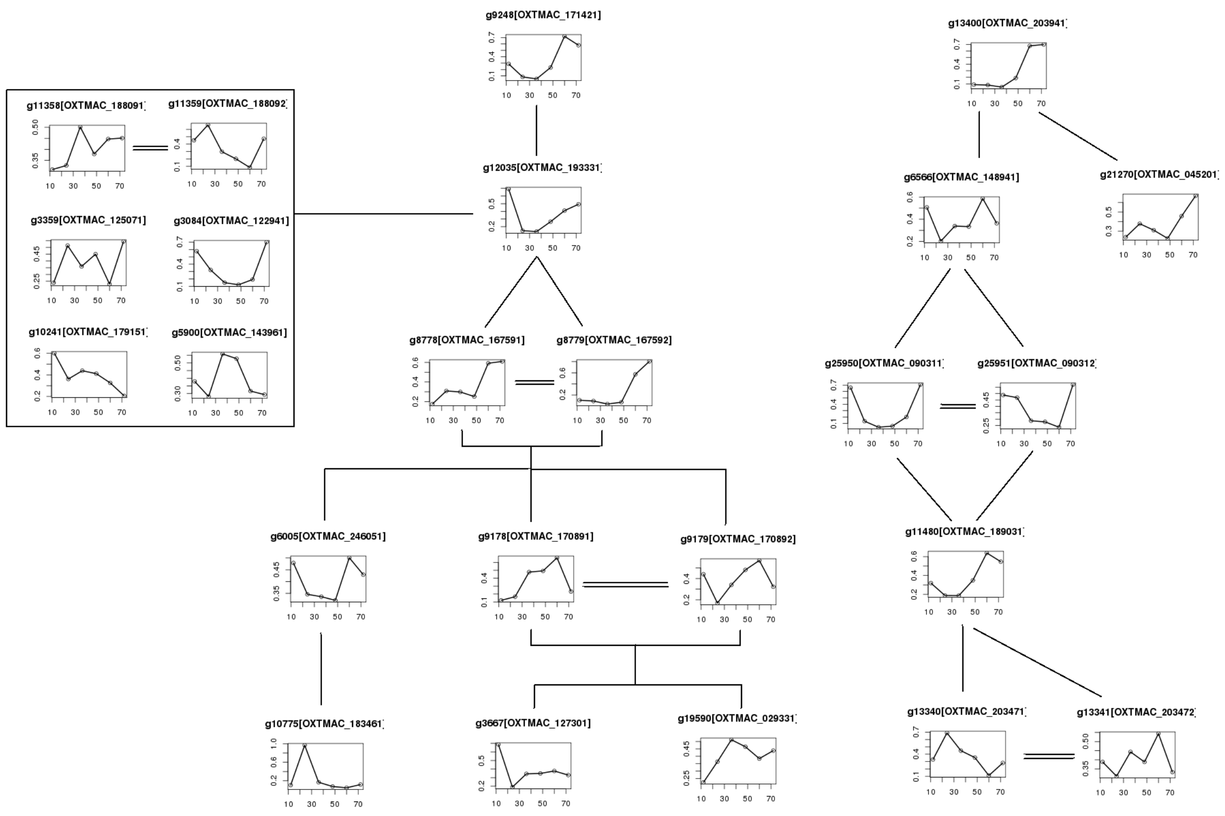

Figure 18. A tree showing the correlations between each IDI 4 contig and its nested genes. The top two genes have an IDI of 4, and vertical edges correspond to genes which are nested within each other. On the left are 6 genes which are nested within one IDI 3 gene, but they themselves have no genes nested within them.

Chapter 5
Conclusion

We were given a data set of gene expression rates of over 27,000 genes, measured across six 12-hour intervals during cellular conjugation. This data was extracted from the ciliate organism *Oxytricha trifallax*, an organism which our lab studies in collaboration with the Landweber Lab in Columbia University. Our goal of this project was to apply various data analysis methods to this data set in the hopes of extracting information about biological processes involved during conjugation. Primarily, we hoped to find out more about what the cell is doing at very specific times during conjugation. Therefore, we looked to clustering to group together genes which were expressing highly at given time points. Using these clusters, we could look at the specific genes within the clusters to determine if anything interesting could be discovered.

Clustering is a mathematical tool for data analysis which has been around for nearly a century, and has seen many applications in various fields. There are many forms of clustering, and no single one is universally better than the others. We analyzed two primary types of clustering: hierarchical clustering and k-means clustering. Hierarchical clustering attempts to gradually cluster points in a data set together based on their proximity to each other. K-means clustering, on the other hand, partitions the data into k clusters to begin with, then attempts to optimize the clusters using the k-means algorithm. For our data. Furthermore, we also looked at TDA as a possible method to look at the general structure of the data. However, the data was too large to perfom TDA on as a whole, and no subsets of the data yielded any relevant information.

After finding our clusters using k-means, we applied GSEA, a statistical test which allowed us to determine that our clusters could not have arisen by simply picking a random sample of our data set. We also ran k-means multiple times using different k values for different numbers of clusters, in the hopes of determining an optimal number of clusters. In the end, we determined that eight clusters was sufficient, because it yielded the primary expression patterns we were looking for, and at this number there were also no repeated patterns.

Gene ontology was then used to determine what biological processes were happening in each cluster of genes. This method compares the genes of our cluster to those genes in more studies organisms, in particular yeast and humans. The output of this analysis was a table of possible protein functions, each with a p-value to determine how likely it is that the genes of the given cluster code for that protein function. A visual representation of this is given in the form of a word graph, as seen in Figure 16.

References

[1] Blast: Basic local alignment search tool. https://blast.ncbi.nlm.nih.gov/Blast.cgi.

[2] Mev: Multiple experiment viewer. http://mev.tm4.org/.

[3] Akman, O., Comar, T., Hrozencik, D., and Gonzales, J. Data clustering and self-organizing maps in biology. *Algebraic and Combinatorial Computational Biology* (2019), 351–374.

[4] Angeleska, A., Jonoska, N., Saito, M., and Landweber, L. Rna-guided dna assembly. *Journal of Theoretical Biology 248* (2007), 706–720.

[5] Ashburner, M., Ball, C., Blake, J., and *et al.* Gene ontology: tool for the unification of biology. *Nat Genet 25* (2000), 25–29.

[6] Bauer, U. Ripser: efficient computation of vietoris-rips persistence barcodes, Aug. 2019. Preprint.

[7] Braun, J., Nabergall, L., Neme, R., Landweber, L., Jonoska, N., and Saito, M. Russian doll genes and complex chromosome rearrangements in oxytricha trifallax. *G3: Genes, Genomes, Genetics 8* (2018), 1669–1674.

[8] Chen, X., Jung, S., Beh, L., Eddy, S., and Landweber, L. Combinatorial dna rearrangement facilitates the origin of new genes in ciliates. *Genome Biology and Evolution 10* (2015), 28592870.

[9] Consortium, T. G. O. The gene ontology resource: 20 years and still going strong. *Nucleic Acids Res* (2019).

[10] Driver, H., and Kroeber, A. Quantitative expression of cultural relationships. *University of California Publications in American Archaeology and Ethnology* (1932), 211–256.

[11] EC, S., JR, B., V, M., P, M., X, C., and et al. The oxytricha trifallax macronuclear genome: A complex eukaryotic genome with 16,000 tiny chromosomes. *PLOS Biology 11* (2013), 1669–1674.

[12] ESTIVILL-CASTRO, V. Why so many clustering algorithms: a position paper. *ACM SIGKDD Explorations Newsletter 4*, 1 (2002), 65–75.

[13] FELLOWS, I. Word clouds. https://cran.r-project.org/web/packages/wordcloud/index.html.

[14] FORGY, E. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics 21* (1965), 768–769.

[15] GHRIST, R. Barcodes: The persistent topology of data. *BULLETIN (New Series) OF THE AMERICAN MATHEMATICAL SOCIETY 45* (02 2008).

[16] LLOYD, S. Least squares quantization in pcm. *IEEE Transaction on Information Theory IT-28 No. 2* (1982), 129–137.

[17] LOVE, M., HUBER, W., AND ANDERS, S. Moderated estimation of fold change and dispersion for rna-seq data with deseq2. *Genome Biol 15* (2014), 550.

[18] NANDA, V., AND SAZDANOVIC, R. Simplicial models and topological inference in biological systems. *Natural Computing Series* (01 2014).

[19] NIELSEN, F. *Hierarchical Clustering*. Springer, 02 2016, pp. 195–211.

[20] OLIPHANT, T. *Guide to NumPy*. 01 2006.

[21] SUBRAMANIAN, A., TAMAYO, P., MOOTHA, V., AND *et al.* Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences 102*, 43 (2005), 15545–15550.

[22] TAUSZ, A., VEJDEMO-JOHANSSON, M., AND ADAMS, H. JavaPlex: A research software package for persistent (co)homology. In *Proceedings of ICMS 2014* (2014), H. Hong and C. Yap, Eds., Lecture Notes in Computer Science 8592, pp. 129–136.

[23] VIRTANEN, P., GOMMERS, R., OLIPHANT, T. E., HABERLAND, M., REDDY, T., COURNAPEAU, D., BUROVSKI, E., PETERSON, P., WECKESSER, W., BRIGHT, J., VAN DER WALT, S. J., BRETT, M., WILSON, J., JARROD MILLMAN, K., MAYOROV, N., NELSON, A. R. J., JONES, E., KERN, R., LARSON, E., CAREY, C., POLAT, İ., FENG, Y., MOORE, E. W., VAND ERPLAS, J., LAXALDE, D., PERKTOLD, J., CIMRMAN, R., HENRIKSEN, I., QUINTERO, E. A., HARRIS, C. R., ARCHIBALD,

A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and Contributors, S. . . SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods 17* (2020), 261–272.

[24] Yerlici, T., and Landweber, L. Programmed genome rearrangements in the ciliate oxytricha. *Microbiology spectrum 2*, 6 (2014).

[25] Zubin, J. A technique for measuring like-mindedness. *The Journal of Abnormal and Social Psychology 33*, 4 (1938), 508–516.

Appendix A

Additional Figures

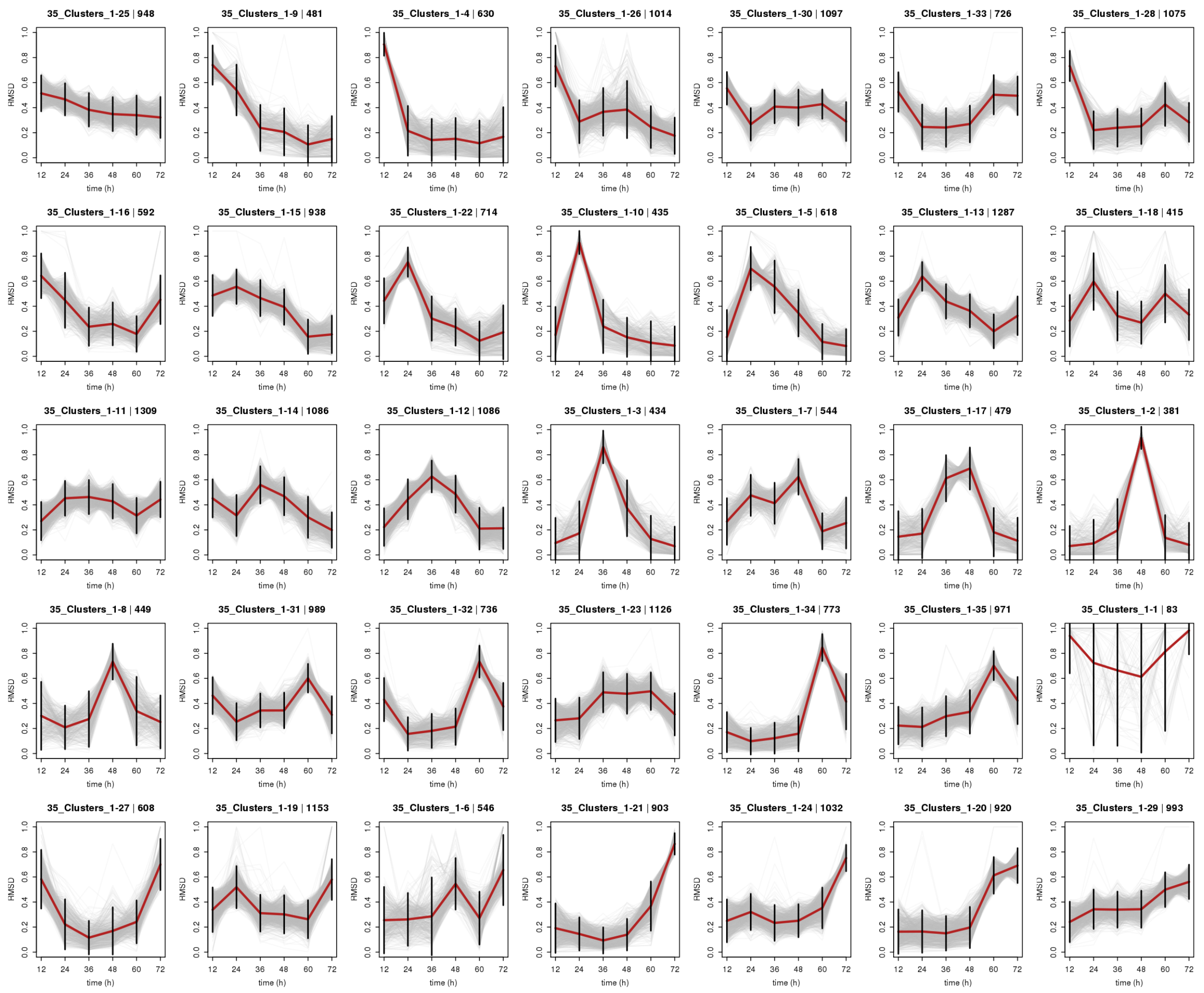

Figure A.01. Cluster plot for 35 clusters. As one can see, there are many shapes present, but many of them are repeated. Hence this was deemed a non-optimal number of clusters.

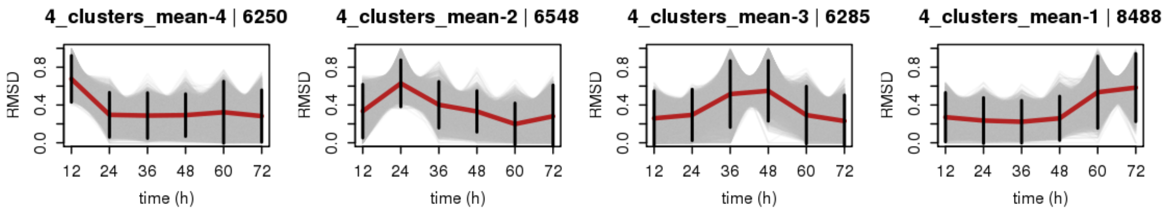

Figure A.02. Cluster plot for 4 clusters. Not enough shapes are present, so this was deemed a non-optimal number of clusters.

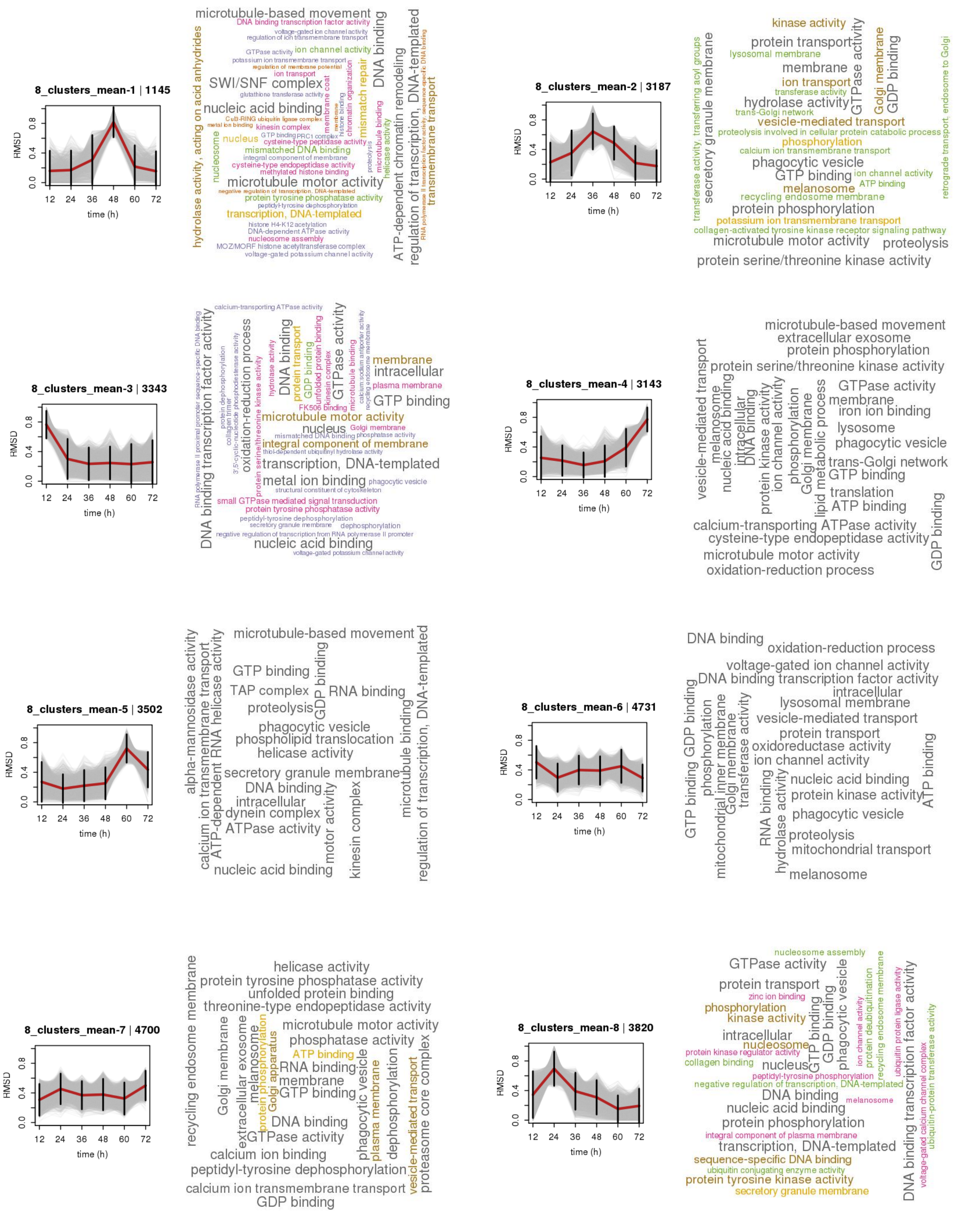

Figure A.03. Word clouds for all 8 clusters seen in Figure 14.