6-24-2014

# Topological Data Analysis of Properties of Four-Regular Rigid Vertex Graphs

Grant Mcneil Conine
*University of South Florida*, gconine@mail.usf.edu

# Topological Data Analysis of Properties of Four-Regular Rigid Vertex Graphs

by

Grant Conine

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

Co-Major Professor: Nataša Jonoska, Ph.D.
Co-Major Professor: Masahiko Saito, Ph.D.
Dmytro Savchuk, Ph.D.

Date of Approval:
June 24, 2014

Keywords: Assembly Graphs, Topological Data Analysis, Homologous Recombination, Mapper

## Dedication

To my wife Heather.

**Acknowledgments**

I owe a great debt of graditude to Dr. Nataša Jonoska and Dr. Masahiko Saito, as well as the rest of the members of their research group at the University of South Florida: Ryan Arredondo, Johnathan Burns, Daniel Cruz, Daria Karpenko, Denys Kukushkin, Maja Milošević, and Rick Wallace.

A further debt is owed to all of the faculty of the Department of Mathematics and Statistics at the University of South Florida. Without their tutelage, guidance, and encouragement, I never would have come so far in my mathematical journey.

This thesis was partially supported by the National Science Foundation under Grant No. DMS-0900671 and the National Institutes of Health under Grant No. R01 GM109459-01. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation or National Institutes of Health.

## Table of Contents

## List of Figures

## Abstract

Homologous DNA recombination and rearrangement has been modeled with a class of four-regular rigid vertex graphs called assembly graphs which can also be represented by double occurrence words. Various invariants have been suggested for these graphs, some based on the structure of the graphs, and some biologically motivated.

In this thesis we use a novel method of data analysis based on a technique known as partial-clustering analysis and an algorithm known as `Mapper` to examine the relationships between these invariants. We introduce some of the basic machinery of topological data analysis, including the construction of simplicial complexes on a data set, clustering analysis, and the workings of the `Mapper` algorithm. We define assembly graphs and three specific invariants of these graphs: assembly number, nesting index, and genus range. We apply `Mapper` to the set of all assembly graphs up to 6 vertices and compare relationships between these three properties. We make several observations based upon the results of the analysis we obtained. We conclude with some suggestions for further research based upon our findings.

# Introduction

One of the most significant problems in modern Bioinformatics and Biomathematics lies not in a lack of interesting experimental data, but rather in the incredibly large quantities of data that even relatively simple biological experiments produce. Thanks to recent advances, it is possible to sequence the entire DNA of individuals, catalog hundreds of proteins, and collect and examine statistical information from thousands of people suffering from diabetes, cancer, and other ailments of our modern age.

The difficulty lies in making sense of the very high-dimensional data that such experiments typically produce. Without careful analysis such data will yield very little in the way of meaningful information. But the human mind is simply not equipped to form intuitions on this sort of data, and the standard statistical toolkit is ill-suited to dealing with such large amounts of data, especially when it is representative of non-linear informations.

Into this gap in our data analysis toolkit steps the "previously-esoteric branch of mathematics" [16] — algebraic topology. Techniques from this branch of mathematics have proven useful in some examples of analyzing large-scale nonlinear data [7, 11, 20].

By construction of a topological structure called a *simplicial complex* over the data set, we can recover some of the topological features of the underlying space, and employ standard techniques to examine these features - the connectedness of the space, tunnels, cavities, and higher-dimensional analogues.

Furthermore we can use clustering analysis to build a visual representation of the data with regards to a chosen partition function. This analysis can reveal relations and structures in data that may otherwise be quite opaque [20].

In particular we will be using these tools to examine a set of graphs closely related the homologous DNA recombination process [1], in the hopes that through coming to an understanding of the features revealed in this relatively well-understood set of data, that we can enable further study of a set of data we do not understand as well: the assembly graphs induced by the DNA recombination process of two model organisms for this process: *Oxytricha trifallax* and *Oxytricha nova*.

In this thesis we first establish the basics of both simplicial analysis and partial clustering analysis (using the tool Mapper [13]), before discussing the derivation of the classes of graphs known as *assembly graphs* that are a model of homologous DNA recombination. Accompanying the discussion of assembly graphs will be the definition of several graph invariants of these graphs. Finally we apply the `Mapper` algorithm to a set of assembly graphs and draw several conjectures based upon the figures generated by this analysis.

# Chapter 1

# Topological Data Analysis

In this chapter we will outline some of the more common tools and techniques used to analyze very large, high dimensional data sets before making a more thorough analysis of the workings of Singh, Mèmoli, and Carlsson's algorithm known as `Mapper` [20] which relies on some of these topological tools.

## 1.1 Connecting data to topology

The first challenge in drawing techniques from topology to bear upon our data analysis is the fact that most topological tools are designed to work upon continuous spaces whereas data consists of a finite set $V$ of discrete points, perhaps (but not necessarily) embedded into the euclidean space $\mathbb{R}^m$. To connect this space of data points to the continuous spaces of topology, we construct a structure know as a *simplicial complex* using the data points as a starting point. The following definitions are drawn from Nanda and Sazdanović's "Simplicial Models and Topological Inference in Biological Systems" [16].

### 1.1.1 Simplicial Complexes

**Definition 1** ([15])**.** A *(abstract) simplicial complex* $(S, \Sigma)$ is a finite vertex set $S$ (called the *base set*) and a collection $\Sigma$ of subsets of $S$ such that if $\tau \in \Sigma$ and $\sigma \subset \tau$, then $\sigma \in \Sigma$.

The simplicial complex made up of every possible subset of $S$ is called the *complete* simplicial complex.

An element $\sigma \in \Sigma$ such that $|\sigma| = n$ is called a *n-simplex* or more generally any $\sigma \in \Sigma$ a *simplex*.

When constructing a simplical complex $\Sigma$ to analyze properties of a dataset $V$, then we require that, for any $v \in V, \{v\} \in \Sigma$.

Often we omit the base set of a simplicial complex when we talk about it, referring to $(S, \Sigma)$ simply as $\Sigma$. If $\sigma, \tau \in \Sigma$ and $\sigma \subset \tau$ we write $\sigma \prec \tau$ and say that $\tau$ is a *face* of $\sigma$.

We note that $S$ is any finite set; it is not necessarily some subset of $\mathbb{R}^n$.

**Definition 2** ([15])**.** Let $\Sigma$ be a simplicial complex, and let $K$ be a subcollection of simplices of $\Sigma$. If $K$ is a simplicial complex in its own right, then we call $K$ a *subcomplex* of $\Sigma$, denoted $K \hookrightarrow \Sigma$.

**Proposition 1** ([15])**.** *If $\Sigma, K$, and $\Lambda$ are simplicial complexes such that $\Sigma \hookrightarrow K, K \hookrightarrow \Lambda$ then $\Sigma \hookrightarrow \Lambda$.*

*Proof.* Follows from definitions. □

**Definition 3** ([15])**.** Let $\Sigma$ be a simplicial complex, and let $N \geq 1$ be a natural number. A *filtration* $\mathscr{F}$ of $\Sigma$ is a collection of subcomplexes $\mathscr{F}_n\Sigma \hookrightarrow \Sigma$ for $n$ in $\{0, \ldots, N\}$ such that

$$\emptyset = \mathscr{F}_0\Sigma \hookrightarrow \mathscr{F}_1\Sigma \hookrightarrow \cdots \hookrightarrow \mathscr{F}_{N-1}\Sigma \hookrightarrow \mathscr{F}_N\Sigma = \Sigma$$

**Definition 4** ([16])**.** Let $N \geq 0$ and $g\colon \Sigma \to \{0, 1, \ldots, N\}$. The *sublevelset of $g$* at the natural number $0 \leq n \leq N$ is

$$S_n(g) := \{\sigma \in \Sigma : g(\sigma) \leq n\}.$$

Thus, any function $f\colon \Sigma \to \mathbb{N}$ (where $\Sigma$ is a simplicial complex) can nearly be used to form a filtration with the sublevelset. We say "nearly" because while it should be clear that $S_n(g) \subset S_{n+1}(g)$ for any $0 \leq n \leq N$, it is not always the case that $S_n(g)$ is a subcomplex of $\Sigma$ for the same $n$. The difficulty arises if $g(\sigma) > n \geq g(\tau)$ with $\sigma \prec \tau$ then $S_n(g)$ contains $\tau$ but not its face $\sigma$. We solve this by restricting our choice of $g$ to functions which avoid this behavior

**Definition 5** ([16])**.** Let $g\colon \Sigma \to \{0, 1, \ldots, N\}$ be a function such that $g(\sigma) \leq g(\tau)$ whenever $\sigma \preceq \tau$. We call g *monotone*.

**Lemma 1** ([16])**.** *Whenever we have a monotone function $g\colon \Sigma \to \{0, 1, \ldots, N\}$, $g$ yields a filtration $\mathcal{S} = \{S_n(g) : 0 \leq n \leq N\}$ of $\Sigma$ which we call the sublevelset filtration of $g$.*

*Proof.* By the above discussion.

**Proposition 2** ([16])**.** *For any filtration $\mathscr{F}$ of a simplicial complex $\Sigma$, there is a unique monotone function $g\colon \Sigma \to \{1, 2, \ldots, N\}$ (for some $N$) so that $\mathscr{F}$ is the sublevelset filtration of $g$.*

*Proof.* Let $g$ be the function which sends $\sigma \in K$ to the smallest $n$ such that $\sigma \in \mathscr{F}_nK$. This $g$ will naturally be monotone, and yields (by definition) a sublevelset filtration identical to $\mathscr{F}$.

Furthermore, given $f, g\colon \Sigma \to \mathbb{N}$, where $f$ and $g$ are monotone, but $f \neq g$ then it must be the case that for some $\sigma \in \Sigma$ that

$$f(\sigma) = n, g(\sigma) = m, (n, m \in \mathbb{N}, n \neq m).$$

Without any loss of generality, assume that $n > m$ which is to say that $f(\sigma) > g(\sigma)$. But then

$$\sigma \in S_m(g) \text{ but, } \sigma \notin S_m(f)$$

Therefore the sublevelset filtrations of $f$ and $g$ must be unique. □

We can see that, given a finite data set $V \subset \mathbb{R}^n$, there are any number of functions that we can choose which will yield a great variety of different filtraions of the complete simplicial complex $\Sigma$ on $V$.

### 1.1.2 Geometric realization

Associated to any simplicial complex $(S, \Sigma)$ is a topological space $|(S, \Sigma)|$ defined in the following manner [7].

Fix dimension $N$ and let $\mathbf{e}_i$ be the $i$-th standard basis vectors for $\mathbb{R}^N$.

**Definition 6** ([14])**.** Let $\mathbf{u} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m\}$ be a collection of $m$ points in $\mathbb{R}^N$. The *convex hull* of $\mathbf{u}$ is given by

$$\text{Conv}(\mathbf{u}) = \{p_1\mathbf{u}_1 + p_2\mathbf{u}_2 + \cdots + p_m\mathbf{u}_m : p_1 + p_2 + \cdots + p_m = 1, p_i \geq 0, (\forall i)\}.$$

It is easy to show that for two points, the convex hull is a straight line segment between the points, and the convex hull of three points yields a triangle with vertices at the three given points. As the dimensions grow, along with the number of points, the exact shape of the convex hull remains analogous.

**Definition 7** ([15, 16])**.** Given a simplicial complex $(S, \Sigma)$ with vertices $S$ ordered such that $S = \{s_1, s_2, \ldots, s_n\}$. This amounts to defining a one-to-one relation between the vertices of $V$ and the standard basis vectors $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n\}$. Define the *geometric realization* $|(S, \Sigma)|$ of the simplicial complex as

$$\bigcup_{\sigma \in \Sigma} |\sigma|$$

where

$$|\sigma| = \text{Conv}\{\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \ldots, \mathbf{e}_{i_m} : \sigma = \{s_{i_1}, s_{i_2}, \ldots, s_{i_m}\}\}$$

and where $\mathbf{e}_j$ is the $j$th standard basis vector.

Thus for any simplicial complex, we can easily and naturally link it with a topological space. This opens up a whole new realm of mathematics which we can use to analyze data and serves as the "root" so to speak of topological data analysis.

### 1.2 `Mapper`

The main tool that we use in this paper is the algorithm developed by Singh, Mèmoli, and Carlsson in their presentation entitled "Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition" [20]. All the definitions are drawn from, or adapted from, that presentation.

`Mapper` takes a large set of, sometimes high-dimensional, data $V$ together with a distance defined on $V$, and from it produces a simplicial complex that represents the data as a simple graph, retaining important topological information like the Betti numbers of the data, and even some geometric information of the original data [20].

`Mapper` works by a process that Singh, Ménoli, and Carlsson in [20] call *partial clustering* which works as follows.

1. Compute the pairwise distance matrix $\mathcal{D} \in \mathbb{R}^n \times \mathbb{R}^n$ (where $|V| = n$) for each pair of data points in $V$.
2. For each $v \in V$, find $f(v)$ (where $f \colon V \to \mathbb{R}$).
3. Create an overlapping partition $\mathcal{P}_f(V) = \{P_{1,f}(V), P_{2,f}(V), \ldots, P_{m,f}(V)\}$ of $V$ by the values of $f(v)$. The exact construction of this partition is determined by some parameters of `Mapper`.
4. Run a *clustering algorithm* on each $P_{i,f}(V)$, returning a collection $\mathcal{N}_i$ of subsets of $P_{i,f}(V)$.
5. Consider $\cup_{i=1}^{m} N_i$. Each subset of this set will be the nodes of the `Mapper` output.
6. Take the intersection graph of $\cup_{i=1}^{m} N_i$. This forms a simplicial complex which is the output of `Mapper`.

#### 1.2.1 Input

The input of the `Mapper` algorithm is a finite data set $V$ which is mapped into some metric space, together with a real-valued function $f \colon V \to \mathbb{R}$ [20]. In addition, `Mapper` takes as input several parameters which effect how the algorithm acts on $V$ and $f$. Some implementations of `Mapper` (such as [13]) allow for further tweaking of these inputs, including pre-filtering data to remove some noise or outliers.

##### Data

As an imput, we can take any finite data set $V$, but we must be able to map $V$ into some metric space. It is important to consider is *which* metric space the data set $V$ is mapped into as the metric space provides a distance function $d(x, y)$ for any $x, y \in V$ which is used throughout `Mapper`.

Of course the Euclidean metric is most commonly used, but there are many other options. For the purposes of our thesis however the Euclidean metric sufficed.

Given two subsets $X, Y \subseteq V$, we can define a number of *component distances* which define a distance between $X$ and $Y$. In the original paper [20] Singh, Mèmoli, and Carlsson use a distance they call *single-linkage clustering* which defines the distance $D_{\min}$ between two clusters $X, Y \subseteq V$ as

$$D_{\min}(X, Y) = \min_{x \in X, y \in Y} d(x, y)$$

Other options include the *complete-linkage clustering* [13]

$$D_{\max}(X, Y) = \max_{x \in X, y \in Y} d(x, y)$$

and the *average linkage clustering* [13]

$$D_{\text{avg}}(X, Y) = \frac{1}{|X| \cdot |Y|} \sum_{x \in X} \sum_{y \in Y} d(x, y).$$

**Partitioning the Data**

There are almost no restriction on the function $f \colon V \to \mathbb{R}$ which is used to partition the dataset $V$, but that does not mean that the choice of $f$ is unimportant. In fact, the choice of a partition function is one of the most important choices to be made in the analysis.

The partition function is used by `Mapper` to divide the data set up into an *overlapping partition* of the data.

**Definition 8.** An *overlapping partition (of a data set $V$) with respect to the function $f$* is a set of ordered, overlapping subsets of the data set such that

$$\mathcal{P}_f(V) = \{f^{-1}(I_k)\}$$

where $I_1, I_2, \ldots, I_n \subset I = [\min(f), \max(f)] \subset \mathbb{R}$ are closed intervals, each $I_k$ is of equal length, $I_i \cap I_{i+1} \neq \emptyset$, and $\cup_k I_k = I$.

The size of each interval and how much each interval overlaps is determined by several settings in `Mapper` which will be discussed later.

Here are a few of the more common choices for the function $f$ as described by Singh, Mèmoli, and Carlsson in [20] given a distance $d(x, y)$ as defined in Subsection 1.2.1, for any $x, y \in V$.

*Density* using the *Gaussian kernel*,

$$f_\varepsilon(x) = C_\varepsilon \sum_{y \in V} \exp\left(\frac{-d(x,y)^2}{\varepsilon}\right)$$

where $\varepsilon > 0$ controls the smoothness of the density function (larger $\varepsilon$ yields a smoother density estimation) and $C_\varepsilon$ is a constant such that

$$\int_{-\infty}^{\infty} f_\varepsilon(x)dx = 1$$

as outlined in [19].

*Eccentricity*, given $1 \leq p < +\infty$,

$$E_p(x) = \left(\frac{\sum_{y \in V} d(x,y)^p}{N}\right)^{\frac{1}{p}}$$

which returns higher values for points lying far from the center of the data, without having to specifically identify a point central to the data.

This can be extended by allowing $p = +\infty$, yielding

$$E_\infty(x) = \max_{x' \in V} d(x,x')$$

which for an $x \in V$ returns the maximum distance from $x$ to any other point in $V$.

*Graph Laplacian*, We create a complete graph $G$ whose vertex set $V$ is the set of all points in the data set $V$. We weight each edge $\{x, y\}$ by

$$w(x,y) = k(d(x,y))$$

where $d(x,y)$ is the distance function of the data set $V$ and $k\colon \mathbb{R} \to \mathbb{R}$ is a "smoothing kernel" [10, 20].

Then the (normalized) graph Laplacian matrix is given by

$$L_{x,y} = \frac{w(x,y)}{\sqrt{\sum_{z \in V} w(x,z)}\sqrt{\sum_{z \in V} w(y,z)}}.$$

The eigenvectors of this normalized Laplacian matrix, among many other interesting features, are a set of orthogonal vectors. The first $i$ eigenvectors can be selected, equally weighted, and used to define a partition function. In practice, this partition function heavily weights sections of the given graph which are connected by many short edges, and lightly weights areas which are connected by only a few edges or by relatively long edges. A full treatment may be found in [10].

The resolution of the algorithm's output determines how fine or coarse the family of overlapping subsets in the partion will be, effecting in turn `Mapper`'s output, and is determined by two variables: the length of the partition intervals and the overlap percentage.

We do not fix length of the partition intervals directly. Instead we adjust a parameter we call the *partition resolution* parameter.

**Definition 9.** The *partition interval* $I$ for a function $f$ on a data set $V$ is defined as

$$I = [\min_{x \in V}(f(x)), \max_{x \in V}(f(x))].$$

We write

$$\ell(I) = \max_{x \in V}(f(x)) - \min_{x \in V}(f(x)).$$

We divide $I$ into overlapping subintervals $I_0, I_1, \ldots, I_m$ where

$$I_i = [a_i, b_i].$$

The *partition resolution* parameter $r$ is a real number such that $0 < r \leq 1$ which controls the length of the partition intervals as follows: For a partition resolution of $r$ and a partition function $f$ we have

$$\ell(I_i) = \ell(I) \cdot r = b_i - a_i, \qquad \text{for } i = 0, 1, \ldots, m-1.$$

The *overlap* parameter $p$ is a real number such that $0 < p < 1$ which defines the portion of partition set $I_i$ which will overlap with its neighbor $I_{i+1}$. That is to say

$$p = \frac{b_i - a_{i+1}}{\ell(I) \cdot r} = \frac{b_i - a_{i+1}}{b_i - a_i}.$$

The overlap parameter determines how likely two nodes will have an edge in between them. While theoretically the overlap could be set to zero, then the `Mapper` output would have no edges at all. Likewise if an overlap greater than 50% is chosen, then the possibility for higher-dimensional complexes to form is introduced, as one data point could appear in three (or even more!) different nodes and thus all three would be joined by edges.

The partition resolution parameter and the overlap parameter together determine the *step length* `Mapper` will take for each partition set.

**Definition 10.** The *step length* $\Delta$ is defined, for a partition resolution parameter $r$, overlap parameter $p$, and partition interval $I$, by

$$\Delta = (1 - p) \cdot \ell(I) \cdot r.$$

Finally we get

$$a_i = min\{I\} + i \cdot \Delta, \qquad \text{for } i = 0, 1, \ldots, m.$$

and

$$b_i = \begin{cases} a_i + \ell(I) \cdot r & \text{for } i = 0, 1, \ldots, m-1, \\ \max f(x) & \text{for } i = m. \end{cases}$$

Then we construct the family of of overlapping partition sets

$$\mathcal{P}^f(V) = \{P_i^f(V) \colon i = 1, 2, \ldots m\}$$

where

$$P_i^f(V) = \{v \colon v \in V, f(v) \in I_i\}.$$

For example, if the partition resolution is set to $r = \frac{1}{8}$ and the overlap percentage is set to $p = .5$, `Mapper` will divide the range of the filtration function into 14 equally sized segments, each of which is $\frac{1}{8}$ of the length of the length of the range of the filtration function $f$, and which all overlap by precisely $50\%$.

### 1.3 Clustering

Once `Mapper` has partitioned the dataset and constructed $\mathcal{P}^f(V)$, it begins "partial clustering" of each set of the partition individually.

*Clustering* is an attempt to group data points together such that points in the same group are in some way "more similar" than points in different groups. Usually (as in our case) that similarity is distance — object in the same group are closer to one another than in other groups.

Clustering is mostly accomplished via a *clustering algorithm*. A clustering algorithm takes the set of data points in question, as well as a measure of the similarity in question (in our case the inter-point distance matrix). Clustering algorithms work in a variety of ways, so we avoid strictly defining a clustering algorithm. The sets of vertices the algorithm creates as it progresses we will refer to as *clusters*. The final cluster that the algorithm outputs (the clusters that appear in the `Mapper` output) we will refer to as *nodes*.

The exact behavior of this clustering depends on the precise clustering algorithm selected. The clustering algorithms we will consider are made up of two parts: a clustering method that determines at what distance two clusters will join together, and a cutoff method which determines when the algorithm should stop clustering and deliver its output.

The only requirements set by Singh, Mèmoli, and Carlsson in [20] of the clustering algorithm used for `Mapper` is

1. The input of the clustering algorithm is the pairwise distance matrix $\mathcal{D} \in \mathbb{R}^n \times \mathbb{R}^n$ where $n$ is the number of data points in a given set of a partition.

2. The clustering algorithm does not specify the number of clusters before running.

Thus we do not consider any clustering algorithms which take anything other than distances between points into account, and we do not specify the number of cluster it will output, giving the algorithm freedom to creates as many or as few clusters as necessary.

The algorithm in `Mapper` creates a series of graphs, $G_j = (V, E_j)$ for $0 \leq j \leq n-1$ where $|E_i| = i$ and $n$ is the number of data points in $V$. Thus $G_0$ is the initial graph and has the data set $V$ as vertices, with an empty edge set. A cluster at a given step $i$ is the vertex set of a connected component of $G_i$.

The clustering algorithm initializes with the set of all data points considered as vertices in graph $G_0$. At step $i+1$ of the algorithm, `Mapper` searches $\mathcal{D}$, the pairwise distance matrix of the data points, for the edge with minimum length which will reduce the number of disconnected components of $G_i$ by one, using the component distance as defined in Subsection 1.2.1. That edge is added to the edge set of $G_i$ to give $G_{i+1}$.

For edge $e$ with endpoints $x, y \in V$, we write $|e| = d(x, y)$.

### 1.3.1 Cutoff method

No matter what component distance is used, the clustering algorithm must also specify a way for the clustering to be halted. In their implementation of the `Mapper` algorithm, outlined in [20], Singh, Mèmoli, and Carlsson use a method described as the *histogram method* to determine when their clustering algorithm should stop adding vertices to a cluster.

We consider $G_{n-1} = G = (V, E)$ from the steps above. We note that $G$ is in fact a tree.

We create $B_1, B_2, \ldots, B_h \subseteq E$ to be subsets of the edge set of $G$ whose contents will be discussed below.

We create a non-overlapping partition of $[0, e_{\max}]$ (where $e_{\max} = \max\{|e| : w \in E\}$)

$$\left(0, \frac{e_{\max}}{h}\right], \left(\frac{e_{\max}}{h}, \frac{2 \cdot e_{\max}}{h}\right], \ldots, \left(\frac{(h-1) \cdot e_{\max}}{h}, e_{\max}\right],$$

where the *histogram cutoff* parameter $h \in \mathbb{Z}$, $h > 0$ determines how many such partitions we will create.

For subset $B_i, e \in E$ we have

$$e \in B_i \Leftrightarrow |e| \in (\eta \cdot (i-1), \eta \cdot i],$$

where the intervals in question are half open intervals and where

$$\eta = \max\{|e| : e \in E\} \cdot \frac{1}{h}.$$

The clustering algorithm then searches $B_1, B_2, \ldots, B_h$ for the $B_i$ with the smallest $i$ such that $B_i = \emptyset$. If such a $B_i$ is found, then the algorithm returns the clusters from the graph

$$G_{FINAL} = \left( V, \bigcup_{j=1}^{i-1} B_j \right)$$

as its nodes. If such a $B_i$ cannot be found, the algorithm returns the clusters from $G_{FINAL} = G$.

By increasing the histogram cutoff parameter we can induce `Mapper` to output more nodes, while decreasing the histogram cutoff will increase the number of nodes we see.

### 1.3.2 Example

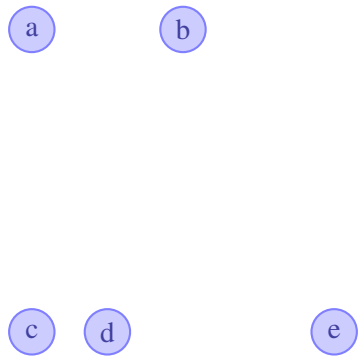



**Figure 1**: The initial data set

**Example 1.** Consider Figure 1 above with five points $a, b, c, d, e$ with the pairwise distance matrix $\mathcal{D}$ given as,

$$D = \begin{pmatrix} 0 & 2 & 4 & \sqrt{17} & 5 \\ 2 & 0 & 2\sqrt{5} & \sqrt{17} & \sqrt{17} \\ 4 & 2\sqrt{5} & 0 & 1 & 3 \\ \sqrt{17} & \sqrt{17} & 1 & 0 & 2 \\ 5 & \sqrt{17} & 3 & 2 & 0 \end{pmatrix}$$

The clustering algorithm begins with $C_0 = \emptyset$ and takes $D$ as input.

1. The algorithm searches $D$ for the edge with the shortest nonzero distance which will reduce the number of disconnected components of $G_0$ by one. It finds that

$$D_{min}(\{c\}, \{d\}) = d(c, d) = 1,$$

and creates a subgraph $A = (\{c, d\}, \{cd\})$ as seen in Figure 2(a). The algorithm sets $G_1 = (V, \{cd\})$.

2. The algorithm searches $D$ for the smallest nonzero distance which will reduce the number of disconnected subgraphs of $G_1$ by one. Remember that using single-linkage clustering, the distance $D_{min}$ between two nodes $X$ and $Y$ is $D_{min}(X, Y) = min\{d(x, y)|x \in X, y \in Y\}$. It finds that,

$$D_{min}(\{a\}, \{b\}) = d(a, b) = 2,$$

and creates subgraph $B = (\{a, b\}, \{ab\})$ as seen in Figure 2(b). The algorithm sets $G_2 = (V, \{ab, cd\})$.

3. The algorithm continues searching $D$ for the smallest nonzero distance which will reduce the number of of disconnected subgraphs of $G_2$ by one. It finds that,

$$D_{min}(A, \{e\}) = d(d, e) = 2,$$

and redefines $A$ as $A = (\{c, d, e\}, \{cd, de\})$, adding $e$ to the vertex set of $A$ and edge $de$ to the edge set of $A$ as seen in Figure 2(c). The algorithm sets $G_3 = (V, \{ab, cd, de\})$.

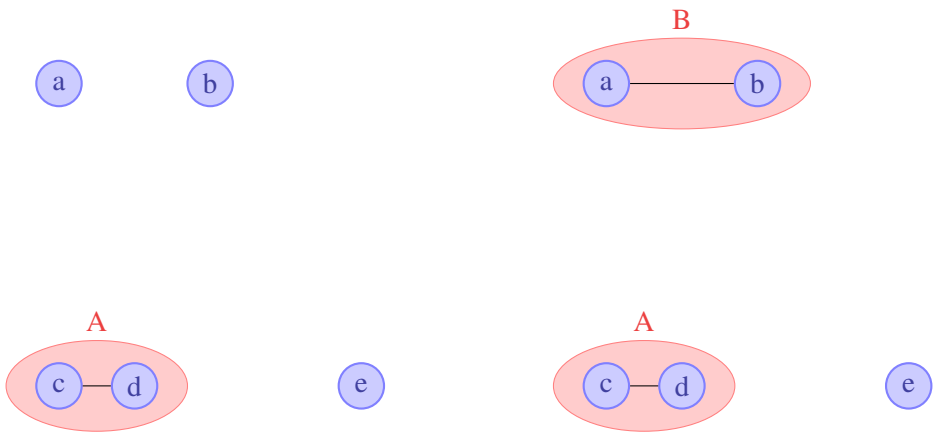
4. The algorithm continues searching $D$ for the smallest nonzero distance which will reduce the number of disconnected subgraphs of $G_3$ by one. It finds that,

$$D_{min}(A, B) = d(a, c) = 4,$$

and redefines $A$ as $A = (\{a, b, c, d, e\}, \{ab, ac, cd, de\})$, adding subgraph $B$ to subgraph $A$ as seen in Figure 2(d). The algorithm sets $G_4 = (V, \{ab, ac, cd, de\})$.

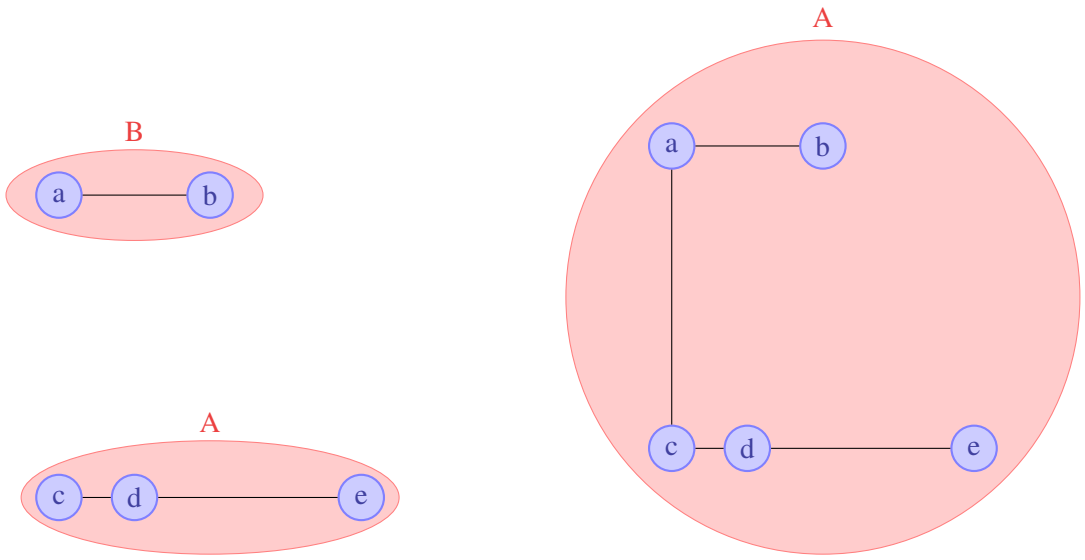
5. As $G_4$ is connected the algorithm renames $G_4 = G$ and halts.

Now $G$ is handed off to the cut-off method which must determine when the clustering ought to have halted.




(a) The data set after step 1

(b) The data set after step 2




(c) The data set after step 3

(d) The data set after step 4

**Figure 2**: The progress of the clustering algorithm

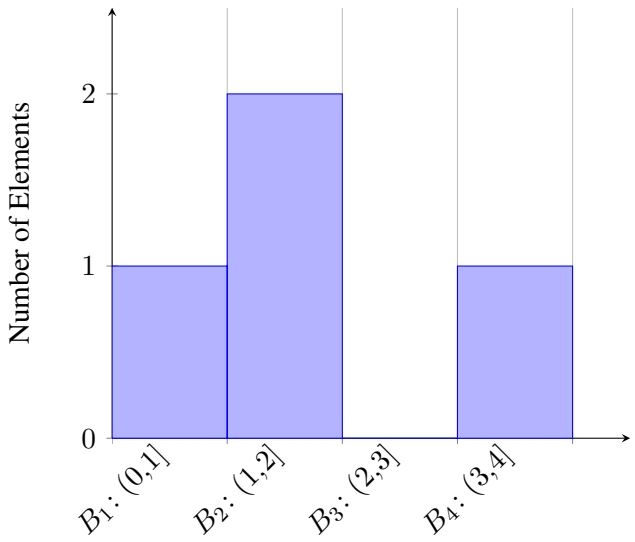



**Figure 3**: An example of a histogram generated to determine clustering cut-off

Suppose that the histogram cut-off was set to 4. Then the cut-off algorithm would divide the edge lengths of $G$ into four intervals: $(0, 1], (1, 2], (2, 3]$, and $(3, 4]$. These entries will serve as the $x$-axis of a histogram, and the number of edges of $G$ falling into each interval will serve as the $y$-axis as seen in Figure 3.

The algorithm recognizes that the interval $(2, 3]$ is empty, and cuts off clustering at a length of 2, i.e. two clusters $X$ and $Y$ will join iff $D_{min}(X, Y) \leq 2$. This will yield a clustering as in Figure 2(c), that is two nodes: $A = \{c, d, e\}$, and $B = \{a, b\}$.

But the choice for the histogram cutoff is very important to get good results as we shall see in the next two examples.

**Example 2.** Suppose that instead of selecting a histogram cutoff of 4 as in Example 1, we selected a smaller histogram cutoff — say two. The resultant histogram can be seen in Figure 4. As we can see there is no empty bars in the histogram, thus the algorithm will cluster every point into a single large node, as in Figure 2(d).

**Example 3.** Likewise supposed we selected a larger histogram cutoff than in Examples 1 and 2. — say 8. The resultant histogram can be seen in Figure 5. As we can see, there is an empty bar straightaways, no edges of a length between 0 and $0.5$, so the algorithm considers there to be no vertices close enough to each other to cluster and the algorithms returns something like Figure 1, where each datapoint is mapped into its own node.

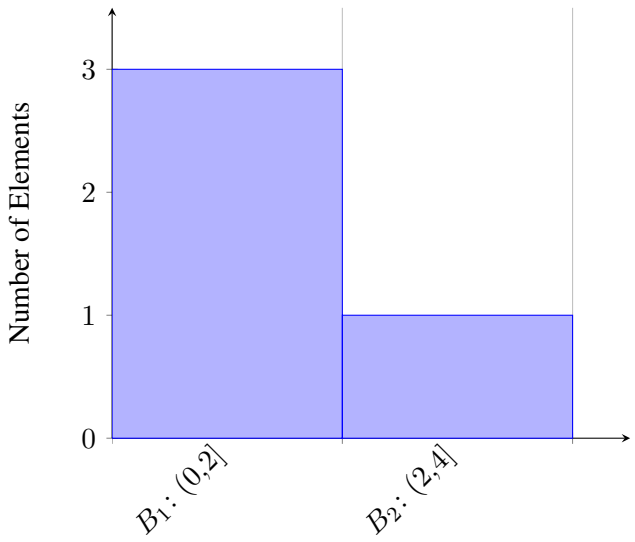



**Figure 4**: An example of a histogram generated to determine clustering cut-off with a histogram cut-off which is set too low

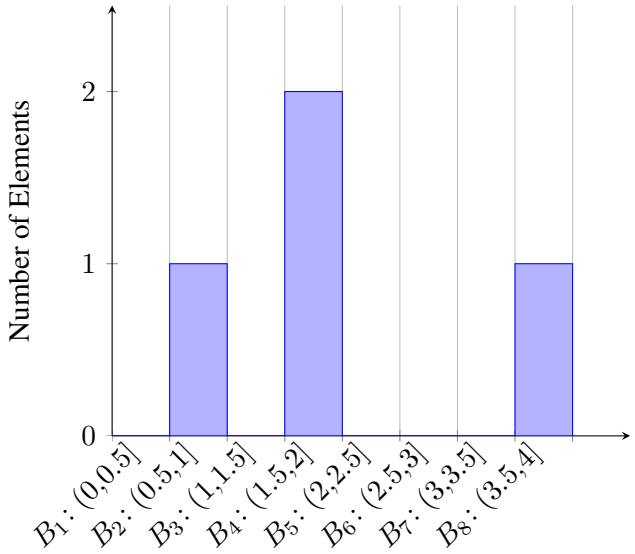



**Figure 5**: An example of a histogram generated to determine clustering cut-off with a histogram cut-off which is set too high

We note that `Mapper` is implicitly utilizing the process of building a simplicial filtration on the set of elements in each partition set when constructing nodes with this clustering algorithm.

### 1.4 `Mapper` Output

`Mapper` takes the union of the nodes from all the clustering algorithms run on each data set as its own node set. The final output is the intersection graph of this node set. We note that two nodes from the same partition set will be disjoint, so there will be no edges between nodes from the same partition set. Instead, as the partition sets overlap with one another, edges will be drawn between nodes from two partition sets which contain overlapping data.

This output is in fact a simplicial complex where the nodes are 0-simplices, edges are 1-simplices, and so on. This is very useful in some cases as we can use many tools from topology to analyze this recovered graph [20].

As a final output, `Mapper` returns three arrays:

1. The *adjacency matrix* $A$ of the output graph. Two nodes $N_i, N_j$ are adjacent if and only if $N_i \cap N_j \neq \emptyset$.
2. An array containing *node info* for every node in the output graph. There are two entries for each node:
   (a) The *filter* entry gives the maximum value of the partition function $f$ for the vertices inside the node described.
   (b) The *set* entry lists the vertices contained within the node described.
3. The *level index* lists which nodes fall into which partition set.

Two nodes $N_i$ and $N_j$ are joined by an edge if $N_i \cap N_j \neq \emptyset$. Thus the $i, j$th entry of $A$ is 1 if $N_i \cap N_j \neq \emptyset$ and 0 otherwise.

This output is in fact a simplicial complex where the nodes are 0-simplices, edges are 1-simplices, and so on. This is very useful in some cases as we can use many tools from topology to analyze this recovered graph [20].

All of this can be used by a program suite such as `Graphviz` to create a graphical representation of the output.

# Chapter 2

# Assembly Graphs

In this chapter we will introduce *assembly graphs* - graphs that we use to model homologous DNA recombination.

As a model organisms for this type of recombination, *Oxytricha trifallax* and *Oxytricha nova* are single celled organisms known as a *ciliate* — named for the small hairs, known as *cilia*, used by the organism to move about. *O. nova* and *O. trifallax* are studied most closely because of the larger, more complex recombinations that their DNA undergo [8, 18]. *O. trifallax* and *O. nova* have two types of internal nuclei — a predominately inactive *micronucleous* and transcriptionally active somatic *macronucleous* [21].

During sexual reproduction, *Oxytricha* exchanges copies of its micronuclei, forms a new diploid micronucleus, and then unscrambles its micronucleus [8], creating a new macronucleus which replaces the original. The micronuclear DNA (hereafter *MIC*) is extremely fragmented, broken down into *macronuclear-destined sequences* or MDS's which might be less than a few hundred base pairs and often appearing out of order or even reversed. Furthermore, up to 90% of the MIC might be *internally eliminated sequences* (IES's) which are discarded during gene unscrambling [4, 21].

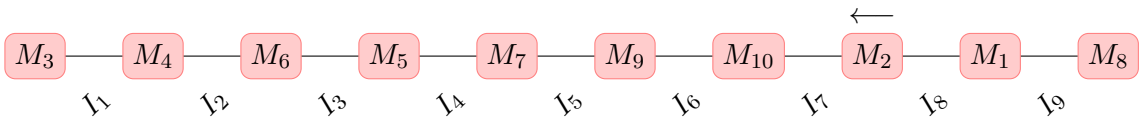
## 2.1 Definitions

In [1] Angeleska, Jonoska, and Saito introduce the concept of *assembly graphs* to model the structure of the MIC during recombination. An example of this is seen in Figure 6.

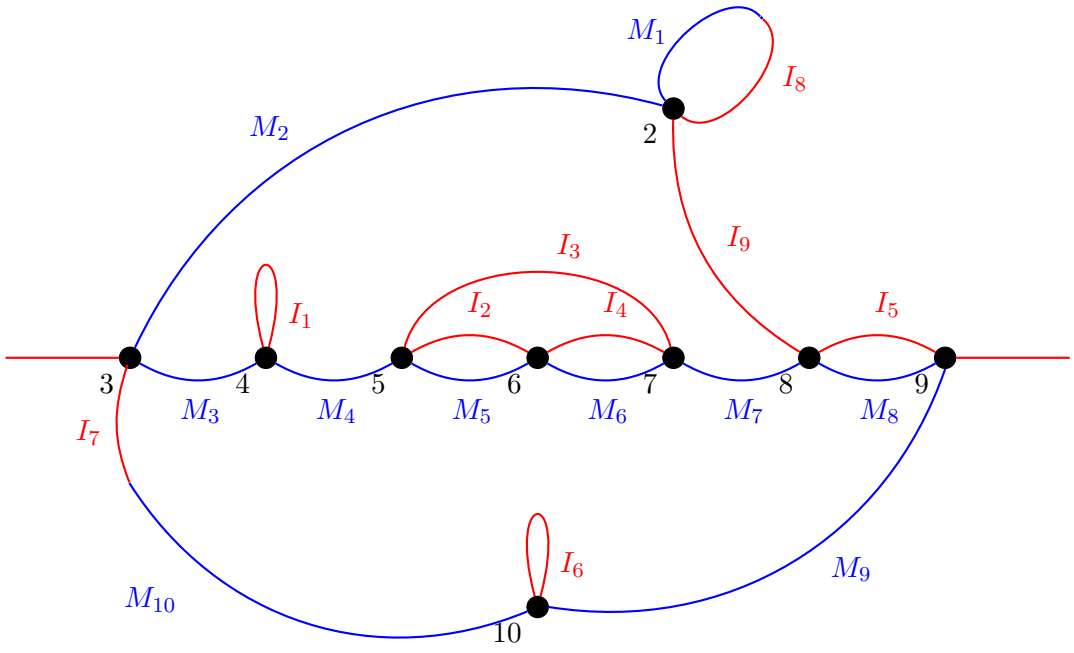
Intuitively, we call a vertex *rigid* if we could represent it as a disk to which each edge is fixed [9].

**Definition 11** ([1, 5].)**.** A vertex $v \in V$ of a graph $\Gamma = (V, E)$ is called *rigid* if we fix an order for the edges of $v$, $\{e_1, e_2, \ldots, e_k\}$ written $(e_1, e_2, \ldots, e_k)$, where consecutive edge $e_i$ and $e_{i+1}$ are neighbors, and only up to a cyclic permutation is allowed in the the order of the edges.

That is to say that a rigid vertex has a prescribed cyclic order in which its edges must appear. If the order of two edges are to be switched, a half-twist must be introduced between those two edges for the graphs to




(a) The scrambled Actin I gene from *Oxytricha trifallax*. Note that vertices labels "$M_i$" means the $i$th MDS and "$I_j$" means the $j$th IES [1, 8, 18]




(b) The unscrambling of Actin I in *Oxytricha trifallax* via an assembly graph [1, 18]

**Figure 6**: The scrambled form and a possible unscrambling process of the Actin I gene from *Oxytricha trifallax* [18]

be equivalent.

**Definition 12** ([1])**.** An *assembly graph* is a finite connected graph, all vertices of which are rigid and are either of degree 4 or degree 1. A vertex of degree 1 is called an *endpoint*, of which there are precisely two.

### 2.2 Assembly Number

In [1], Angeleska, Jonoska, and Saito introduce the concept of the *assembly number* of assembly graphs, which in this context tells whether or not a given assembly graph can represent the unscrambling of a single gene.

**Definition 13** ([1])**.** Let $\Gamma = (V, E)$ be an assembly graph. An *open path* $\gamma$ in $\Gamma$ is a homeomorphic image of the open interval $(0, 1)$ in $\Gamma$. An open path may also be represented as a sequence

$$(e_1/v_0), v_1, e_2, v_2, e_3, \ldots, v_{k-1}, e_k, v_k, (e_{k+1}/v_{k+1})$$

where $v_1, v_2, \ldots, v_k$ are vertices in $\Gamma$ with $v_i \neq v_j$ for $i \neq j$ and $e_1, e_2, \ldots, e_k, e_{k+1}$ are edges in $\Gamma$ with $v_{i-1}$ and $v_i$ the endpoints of $e_i$, excluding the initial vertex of $e_1$ and the terminal vertex of $e_{k+1}$.

Two open paths are *disjoint* if they share no vertices in common.

**Definition 14** ([1])**.** A set of pairwise disjoin open paths in $\Gamma = (V, E)$ is called *Hamiltonian* if their union contains all 4-valent vertices of $\Gamma$.

A single path $\gamma$ in graph $\Gamma$ is called Hamiltonian if $\{\gamma\}$ is a Hamiltonian set in $\Gamma$.

**Definition 15** ([1])**.** A *polygonal path* in graph $\Gamma = (V, E)$ is an open path $\gamma = ((e_1/v_0), v_1, e_2, v_2, e_3, \ldots, v_{k-1}, e_k, v_k, (e_{k+1}/v_{k+1}))$ such that $e_i$ and $e_{i+1}$ are neighbors for all $i$.

That is to say that in a 4-regular graph, a polygonal path is an open path which takes either a "left" or "right" turn at each vertex — it never passes straight through a vertex.

**Definition 16** ([1])**.** The *assembly number* of assembly graph $\Gamma = (V, E)$, written $\mathrm{AN}(\Gamma)$, is defined as

$$\mathrm{AN}(\Gamma) = \min\{k : \text{there exists a Hamiltonian set of polygonal paths } \{\gamma_1, \gamma_2, \ldots, \gamma_k\} \text{ in } \Gamma\}.$$

**Definition 17** ([1])**.** A *transverse path* in a graph $\Gamma = (V, E)$ is an open path $\gamma = (v_0, e_1, v_1, e_2, v_2, e_3, \ldots, e_n, v_n)$ if $v_0, v_n$ are endpoints of $\Gamma$, or $\gamma = (v_0, e_1, v_1, e_2, v_2, e_3, \ldots, e_n)$ if one of $e_n$'s endpoints is $v_0$, with the following properties:

1. $v_0, v_1, \ldots, v_n$ are an ordering of a subset of $V(\Gamma)$ with the same vertex appearing at most twice,
2. $\{e_1, e_2, \ldots, e_n\} \subseteq E(\Gamma)$ are a set of unique edges of $\Gamma$, and
3. The edges $e_i$ and $e_{i+1}$ ($i = 1, 2, \ldots, n-1$) are not neighbors at the vertex $v_i$ (which is to say that the path traced through $v_i$ by $\gamma$ takes neither a left nor a right turn at $v_i$).

**Definition 18** ([1])**.** An assembly graph containing a transverse Eulerian path (i.e. contains a transverse path which visits each edge of $\Gamma$ precisely once) is called a *simple* assembly graph.

Hereafter all assembly graphs are assumed to be simple assembly graphs.

**Definition 19** ([1])**.** For $n \in \mathbb{R}, n > 0$, the *minimum realization number* of $n$ is defined as

$$R_{\min}(n) = \min\{|\Gamma| : \mathrm{AN}(\Gamma) = n\}$$

where $\Gamma$ is an assembly graph, and $|\Gamma|$ is the number of vertices in $\Gamma$.

It has been observed in [1] that $R_{\min}(1) = 1, R_{\min}(2) = 4$. It was also proven in [1] that $R_{\min}(n) \leq 3(n-1)+1$ for $n > 0$ and further observation in [6] showed that $R_{\min}(3) = 7$.

### 2.3 Double Occurrence Words

There is also a close association between the structure of MIC recombination and a set of words called *double occurrence words*.

**Definition 20** ([17])**.** An *alphabet* $\Sigma$ is a finite set of elements, called *letters*.

**Definition 21** ([17])**.** A *word* (over an *alphabet* $\Sigma$) is a series of *letters* $a_i \in \Sigma$.

We define $\epsilon$ to be the *empty word*.

**Definition 22** ([2, 17])**.** A *double occurrence word* (DOW) $w$ is a word with letters from a finite alphabet $\Sigma$, such that every letter in $w$ appears precisely twice.

If we label the MDS's in the order they appear in the MAC ($\mathrm{MDS}_0$ through $\mathrm{MDS}_{k+1}$), and then associating a letter $a_i$ with the end of $\mathrm{MDS}_i$ and the beginning of $\mathrm{MDS}_{i+1}$, then a DOW is formed over the alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$ by taking the letters in the order they appear in the MIC.

There is also a natural relation between double occurrence words and assembly graphs. We may make an assembly graph for any double occurrence word by naming the vertices of the graph after the letters in the double occurrence word, and drawing edges between those vertices/letters which appear concurrently in the double occurrence word. See Figure 7 for two simple examples.

### 2.4 Nesting Index

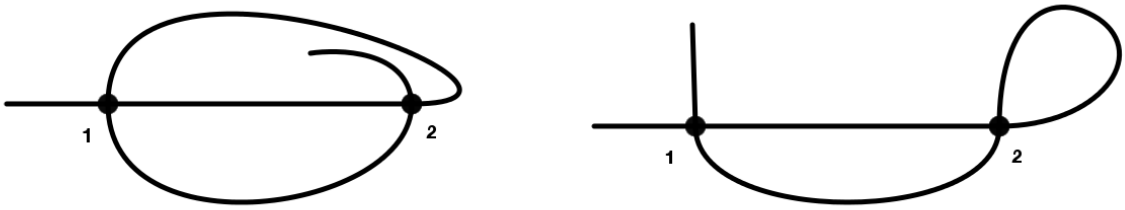
In [2], Arredondo gives an invariant on double occurrence words called the *nesting index* of a word. It is motivated by patterns of recombination that are often seen in DNA rearrangement of *O. trifallax* and *O. nova* in laboratory settings.

**Definition 23** ([2])**.** A *return* word is a word of the form

$$a_1 a_2 \cdots a_n a_n \cdots a_2 a_1 \qquad \text{for } a_i \in \Sigma \text{ and } a_i \neq a_j \text{ for } i \neq j.$$

A *repeat* word is a word of the form

$$b_1 b_2 \cdots b_n b_1 b_2 \cdots b_n \qquad \text{for } b_i \in \Sigma \text{ and } b_i \neq b_j \text{ for } i \neq j.$$




(a) The assembly graph of repeat word 1212 (b) The assembly graph of return word 1221

**Figure 7**: Assembly graphs of repeat and return words

**Definition 24** ([2])**.** We say that $u$ is a *subword* of $w$ ($u \sqsubseteq w$) if $w = v_1 u v_2$ for some words $v_1, v_2$ over $\Sigma$ (where $v_1$ or $v_2$ might possibly be the empty word $\epsilon$).

**Definition 25** ([2])**.** If $w = v_1 u v_1$, the we write $w - u = v_1 v_2$.

**Definition 26** ([2])**.** We say that a word $w'$ is a *maximal* return or repeat subword of $w$ if for any return or repeat word $u \sqsubseteq w$, either $|u| = |w'|$ or $u = w$.

**Definition 27** ([2])**.** We say that $w'$ is obtained from $w$ by a *reduction operation* if either,

1. $w' = w - \{u : u \text{ is a maximal return or repeat subword of } w\}$, or
2. If for some $a \in \Sigma$, $w' = w - a$.

**Definition 28** ([2])**.** The sequence of words $(u_0, u_1, \ldots, u_k)$ is a *reduction* on the word $w$ if

1. $u_0 = w$,
2. $u_{i+1}$ is obtained from $u_i$ by a reduction operation, for $0 \leq i \leq k$, and

3. $u_k = \epsilon$.

**Definition 29** ([2])**.** The *nesting index* of a double occurrence word $w$, written $\mathrm{NI}(w)$, is defined as

$$\mathrm{NI}(w) = \min\{k : (u_1, u_2, \ldots, u_k) \text{ is a reduction of } w\}.$$

Additionally in [2] it is observed that the shortest assembly word $w$ such that $\mathrm{NI}(w) = 4$ is on 5 letters, and that the maximum nesting index attained for assembly words on 5 and 6 letters is in fact four.

### 2.5 Genus Range

Another method of measuring the complexity of assembly graphs was presented by Buck et al. in [5]. There the authors present the concept of the *genus range* of assembly graphs, as well as some preliminary facts about the genera of graphs.

Arredondo in [3] showed that there is no direct relation between nesting index and the genus range (as it is possible to have an assembly graph with arbitrarily large nesting index and arbitrarily large genus), it is still highly likely that as both measure the complexity of the structures of assembly graphs, that there may be some correlation between the two.

**Definition 30** ([5])**.** An embedding of an assembly graph $G$ into a surface S is *cellular* if every component of the complement of the union of the vertices and edges is equivalent to an open disk in the surface.

We want to investigate what kinds of surfaces a given assembly graph may be cellularly embedded into. A common topological method of distinguishing surfaces is by the *genus* of the surface.

The genus of an orientable surface is a well-known topological concept. Intuitively, the genus of a given surface is a measure of how many holes there are in the given surface. More specifically, it is the measure of how many cuts may be made in a surface without rendering that surface disconnected [14].

**Definition 31** ([5])**.** The *minimum orientable genus* of a graph $\Gamma$, denoted $g_{\min}(\Gamma)$, is the smallest number $n \in \mathbb{Z}$, $n \geq 0$ such that $\Gamma$ can be cellularly embedded into an orientable surface of genus $n$ [5].

Likewise, the *maximum orientable genus* of a graph $\Gamma$, denoted $g_{\max}(\Gamma)$, is the largest number $N \in \mathbb{Z}$, $N \geq 0$ such that $\Gamma$ can be cellularly embedded into an orientable surface of genus $N$.

**Definition 32** ([5])**.** The *genus range* of an assembly graph $\Gamma$, denoted $\mathrm{GR}(\Gamma)$ is defined as

$$\mathrm{GR}(\Gamma) = \{n : \Gamma \text{ may be cellularly embedded into an orientable surface of genus } n\}.$$

Calculating the genus range of a 4-regular graph is a matter of constructing a family of surfaces which the given graph can cellularly embedded into. This may be done by a construction called the *ribbon graph* construction, as follows:

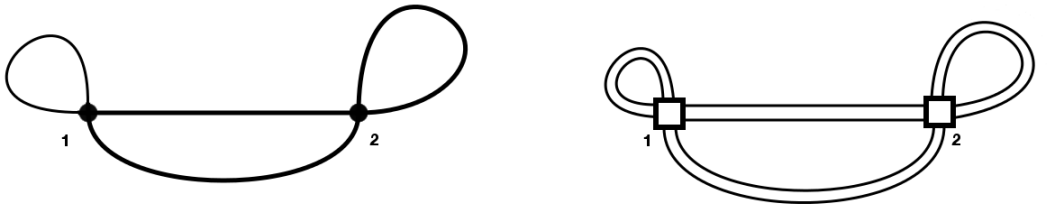
Let $\Gamma$ be an assembly graph with $n$ vertices, represented by a double-occurence word $w$. We identify the endpoints of $\Gamma$ with one another, creating a rigid-vertex, 4-regular graph.

1. For each vertex $i, i = 1, 2, \ldots, n$ associate $i$ with a square, each side of which is incident to an edge, such that neighboring edges are incident to neighboring sides of the square.

2. For each edge $e$ with endpoints $i$ and $j$ ($i, j = 1, 2, \ldots, n$), we associate $e$ with a band (or ribbon) in such a way that the resultant surface $F$ is orientable. An example of the results of these first three steps may be found in Figure 8.

3. Now that an orientable surface has been obtained, we can determine the *Eulerian characteristic* $\chi$ of the surface. It is well known that for an compact orientable surface $F$ of genus $g(F)$ and $b(F)$ boundary components, $\chi(F) = 2 - 2g(F) - b(F)$. As $F$ is homotopic to $\Gamma$, we may determine that $\chi(F) = \chi(\Gamma) = -n$ where $n$ is the number of vertices in $\Gamma$. Thus $g(F) = (1/2)(n - b(F) + 2)$.

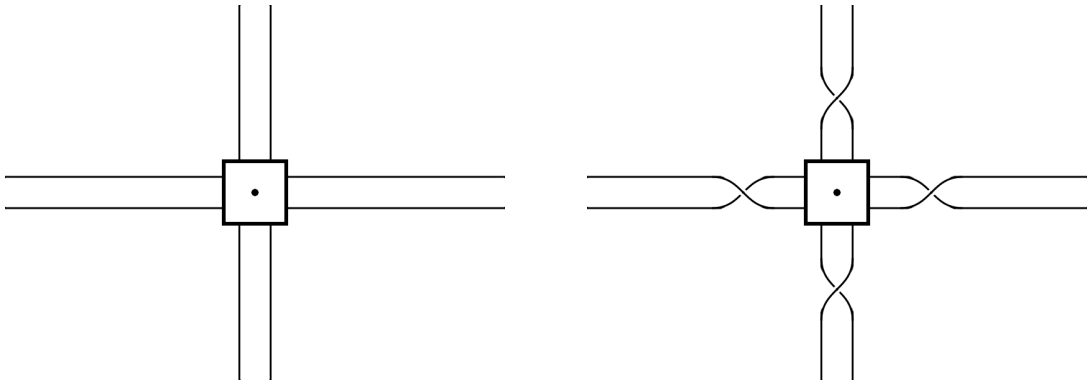
4. To find the genus *range* one has to consider that each square which correlates to a vertex may be oriented either right-ways up or upside down. Orienting a square upside down introduce a half-twist along each band which is incident to it. An example of this process may be found in Figure 9. There are in fact $n^2$ different ribbon graphs for a graph $\Gamma$. Counting the genus of all of these ribbon graphs yields the genus range of $\Gamma$.




(a) The assembly graph of the word 1221

(b) A ribbon graph representation of the assembly graph of 1221

**Figure 8**: An example of a ribbon graph construction



(a) A vertex and incident edges of a ribbon graph without a half-twist

(b) A vertex and incident edges of a ribbon graph with a half-twist

**Figure 9**: The introduction of a half twist at a vertex in a ribbon graph

In [5], Buck et al. proved that given an assembly graph $\Gamma$ and $g_{\min}(\Gamma) = n$, and $g_{\max}(\Gamma) = N$, $(n \leq N)$ that for any $i$, $n \leq i \leq N$, it is possible to embed $\Gamma$ into a surface of genus $i$. Thus we can express the genus range of a graph as the the interval $[g_{\min}(\Gamma), g_{\max}(\Gamma)]$. Furthermore, there is a natural ordering of the genus range by ordering the genus ranges lexicographically, first by maximum genus and then by minimum genus. That is to say that

$$\begin{aligned}\min(\Gamma), \max(\Gamma)] <& [\min(\Lambda), \max(\Lambda)] \\ &\Leftrightarrow \max(\Gamma) < \max(\Lambda) \text{ or,} \\ &\Leftrightarrow \max(\Gamma) = \max(\Lambda) \text{ and } \min(\Gamma) \leq \min(\Lambda).\end{aligned}$$

# Chapter 3

# Input Configuration and Results

With the preliminaries addressed, we turn to our implementation of `Mapper` and the particular data set we ran it on.

We ran a version of the `Mapper` algorithm written for MATLAB ([12]), following the algorithm in [20].

## 3.1 Data

The data used was drawn from a database maintained by the research group of Drs. Nataša Jonoska and Masahiko Saito at `http://knot.math.usf.edu`. We chose to represent every double occurrence word (up to 6 letters) by a point in $\mathbb{R}^3$ where the $x$-value is given by the assembly number of the related assembly graph ranging from 1 to 2, the $y$-value is given by the nesting index of the related assembly graph ranging from 1 to 4, and the $z$-value is given by the lexicographic order of the genus range of the related assembly graph ranging from 1 to 10. We have explained the limitations of the assembly number and nesting index. As examination of Figure 10 shows, the greatest value attained by the ordered genus range is $[3, 3]$, which translates to $10$. There are a total of $11,464$ different assembly graphs considered from the table.

The data will be presented in this paper in the following form:

$$(\textit{Assembly Number}, \textit{Nesting Index}, [\textit{Minimum Genus}, \textit{Maximum Genus}]),$$

with the genus repeated if the minimum and maximum genus are the same.

The assembly word data was obtained by Jonathan Burns in conjunction with the research for [6]. The nesting index data was obtained by Ryan Arredendo in conjunction with research for [3]. The genus range data was originally obtained by Egor Dolzhenko in the course of his research for [5]; the version of the data used for this paper was derived by Ryan Arredendo in conjunction with research for [2]. All relevant data is available at `http://knot.math.usf.edu/data`.

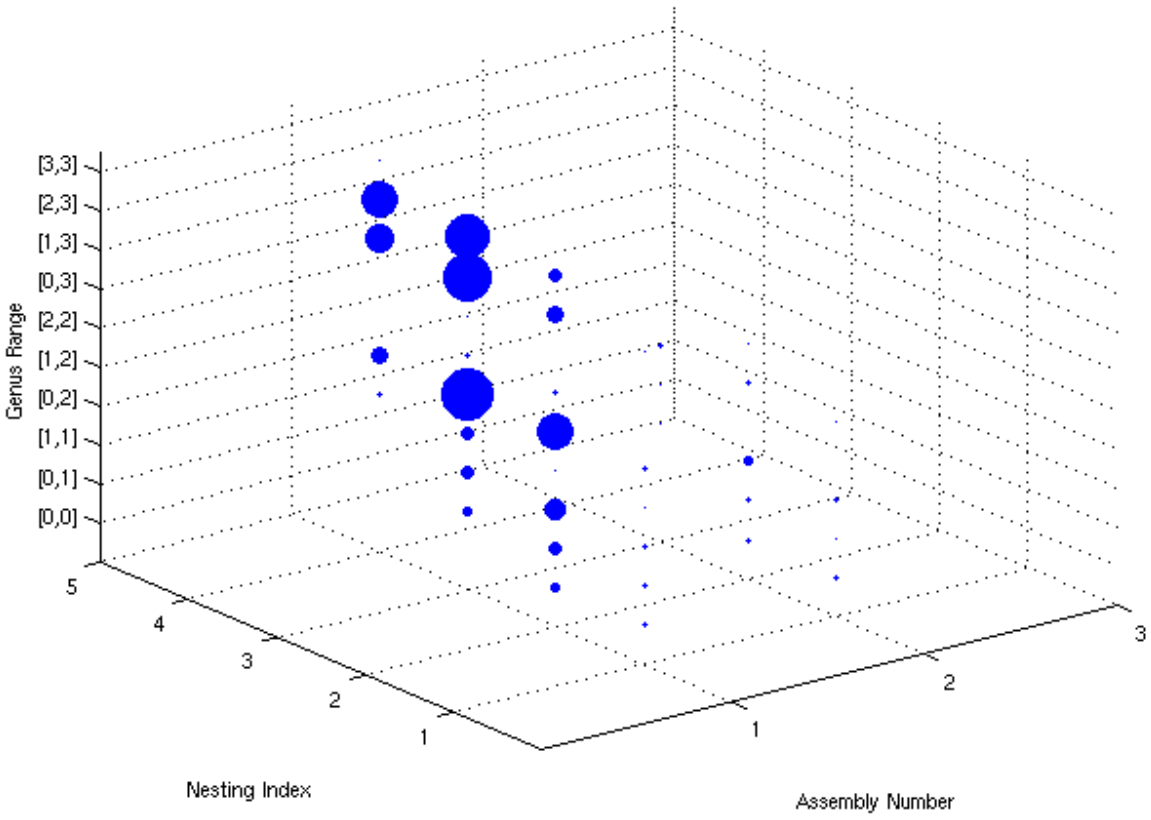



**Figure 10**: The data, rendered in 3D, with vertex size determined by the number of data points which appear there

All of the data was compiled in a comma-seperated value database and loaded into MATLAB ([12]). MATLAB was used to calculate the pairwise Euclidean distance matrix of all of the data points in the normal fashion.

### 3.2 `Mapper` configuration

`Mapper` was configured to use the Euclidean distance from the first word ("11") as the partition function. The pairwise distance matrix was computed separately using MATLAB ([12]) to speed computation.

After running several different configurations as seen in the Appendix, a pattern began to emerge in our outputs. In the interest of completeness, and in the hope that the reader finds them informative and elucidating, these extra outputs are included in the Appendix. We will concentrate the rest of our investigation on

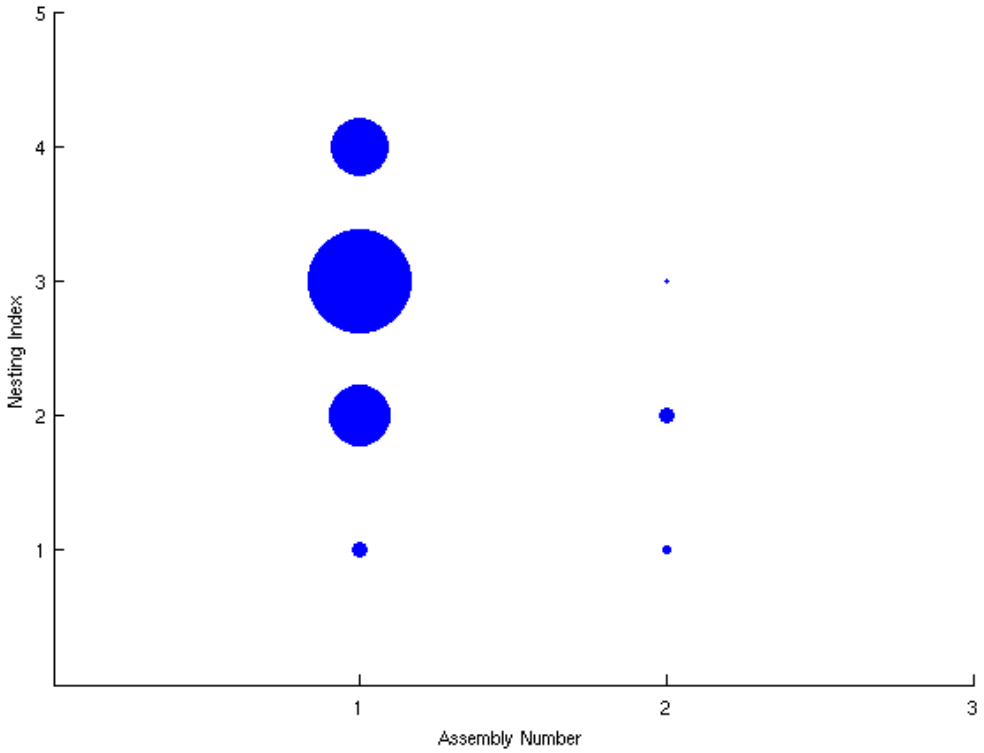



**Figure 11**: 2 dimensional projections of the data set: Assembly Number by Nesting Index

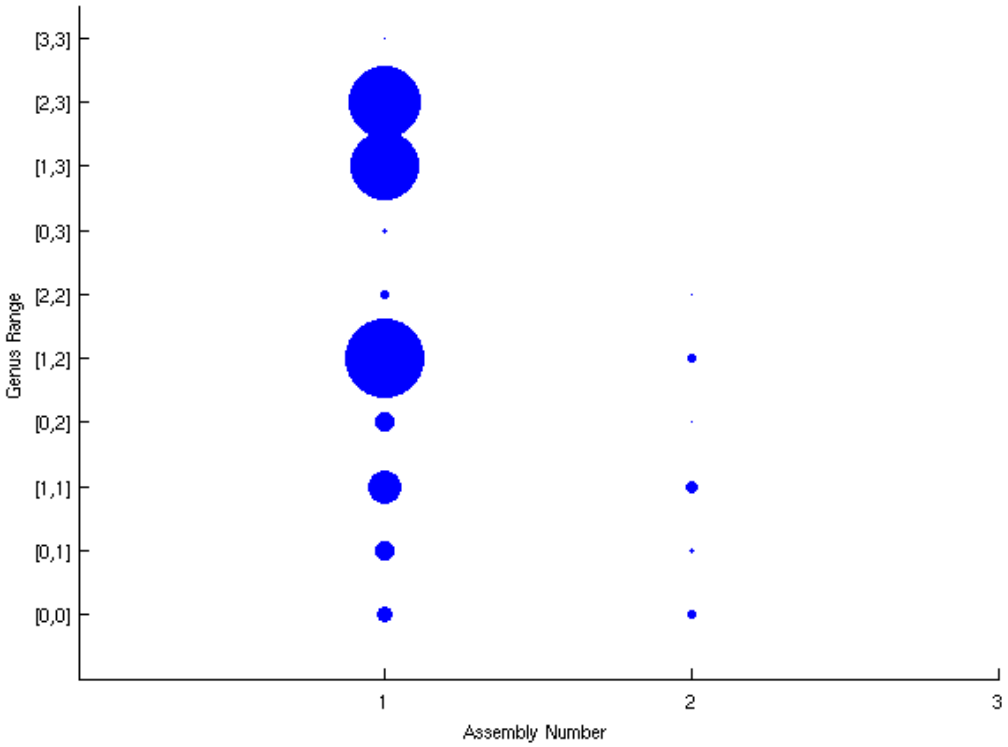



**Figure 12**: 2 dimensional projections of the data set: Assembly Number by Genus Range

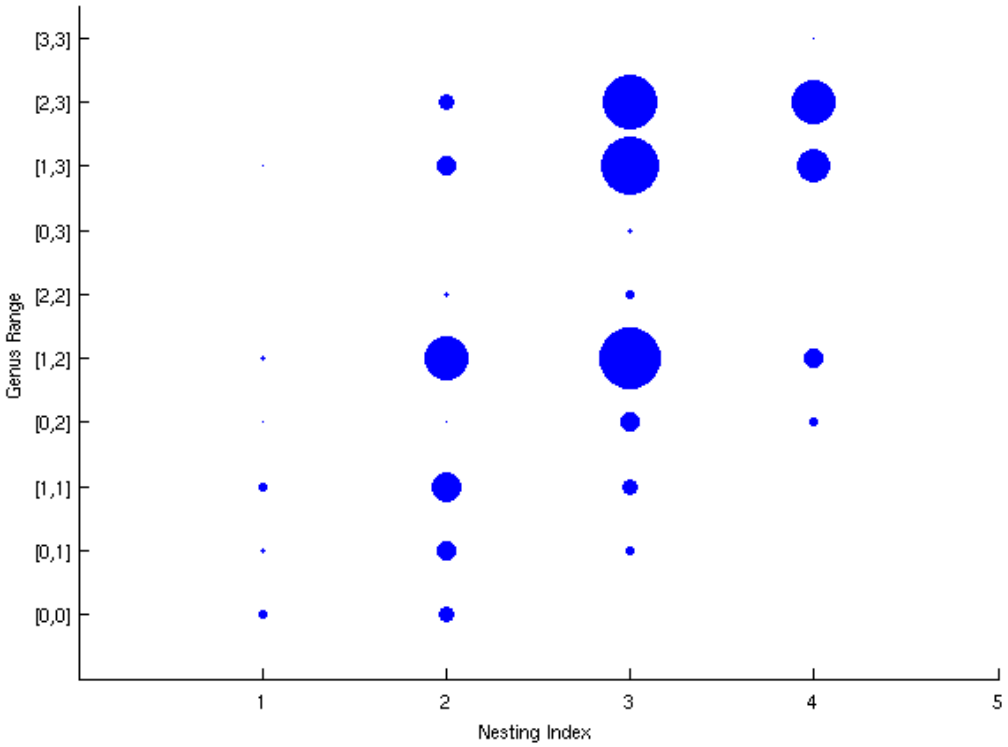



**Figure 13**: 2 dimensional projections of the data set: Nesting Index by Genus Range

Figure 14 which is fairly representative of the other outputs.

To obtain the results seen in Figure 14 we set `Mapper` to use a histogram cut-off parameter $h = 13$ (we should note that a cut-off parameter of 14 achieves an identical graph), as well as the default partition resolution of $r = \frac{1}{8}$ and and overlap parameter $p = .5$. The larger-than-usual histogram cut-off parameter was selected to draw out more features, at the cost of generating a bit more noise. The features we will examine may be found in any of the outputs we obtained, but are sometimes obscured or joined together by a tenuous link that suggests the adjacency is due to improper settings rather than an underlying feature.

The output was rendered by the `neato` package of `graphviz`, a package which seeks to avoid overlapping edges while drawing the shortest possible edges — a "spring model" if you will.

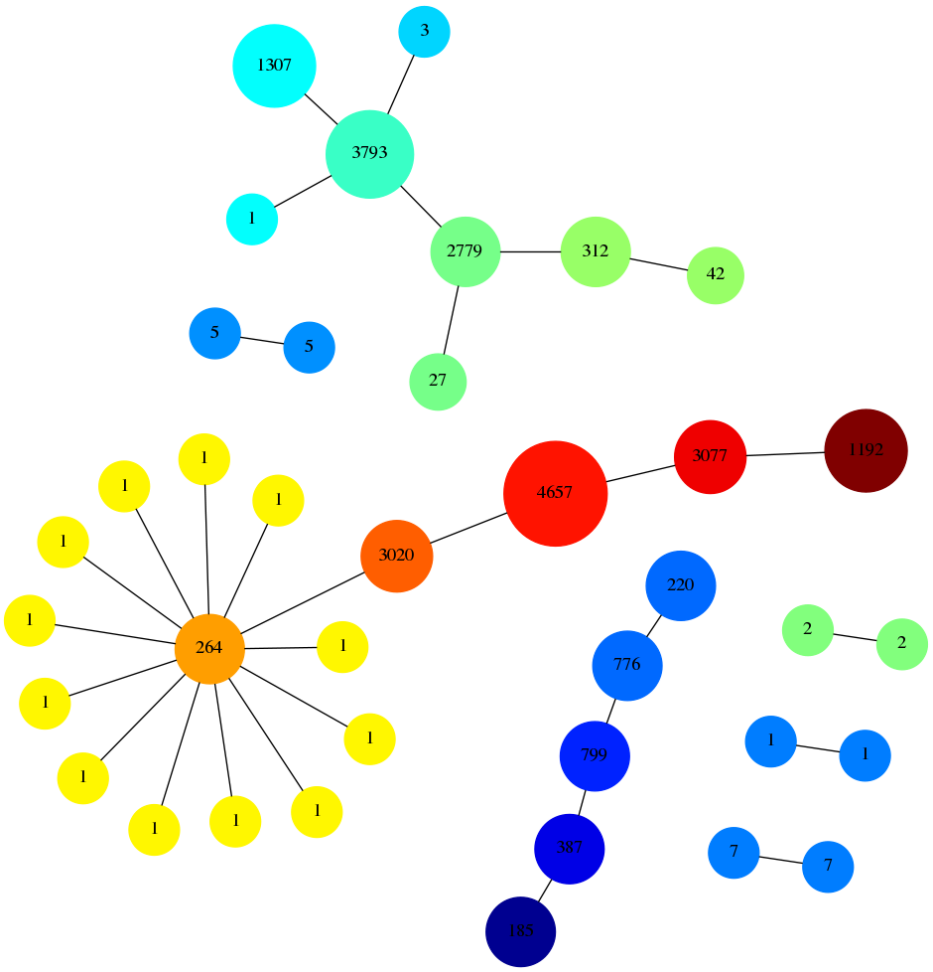



**Figure 14**: Output of `Mapper` with histogram cutoff parameter $h = 13$, partition resolution $r = \frac{1}{8}$, and overlap parameter $p = .5$

### 3.3 Results

Here we present the output resulting from these parameters, and categorize the three main features we find in the output.

We have named the three main features of the output: the Line, the Flower, and the Stick Figure. We will outline the contents of each and some observations below, as well as a few words on the several "noisy" clusters that appear as well.

We should note here that every graph structure in the data set appears in two nodes as each graph (excepting those which appear very near to $\min(f)$ and $\max(f)$) will appear in two partition sets. Nodes are colored by the average filter function value of their constituent data points, colored from dark blue (relatively low average value of $f$) to yellow (relatively high average value of $f$). The nodes are marked with the number of data points which they contain.

#### 3.3.1 The Line

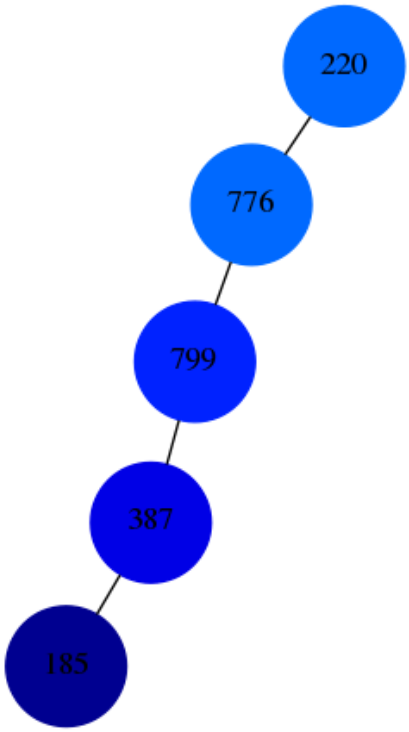



**Figure 15**: The Line feature from the `Mapper` output

Figure 15 is the feature we have named the Line. This feature contains the simplest assembly graphs, and the closest relation between assembly number and genus range and nesting index of our data set, and is the simplest cluster in structure. Moving from left to right (and bottom to top):

Line Node 1

Contains vertices $(1, 1, [0, 0])$, $(1, 2, [0, 0])$, $(1, 1, [0, 1])$, and $(2, 1, [0, 0])$.

Line Node 2

Contains vertices $(1, 1, [0, 1])$, $(1, 2, [0, 0])$, $(1, 2, [0, 1])$, $(2, 1, [0, 0])$, $(2, 1, [0, 1])$, $(2, 2, [0, 0])$, and $(2, 2, [0, 1])$.

Line Node 3

Contains vertices $(1, 1, [1, 1])$, $(1, 2, [0, 1])$, $(1, 2, [1, 1])$, $(1, 3, [0, 1])$, $(2, 1, [0, 1])$, $(2, 2, [0, 0])$, and $(2, 2, [0, 1])$.

Line Node 4

Contains vertices $(1, 1, [1, 1])$, $(1, 2, [1, 1])$, $(1, 3, [0, 1])$, $(1, 3, [1, 1])$, $(2, 1, [1, 1])$, and $(2, 2, [1, 1])$.

Line Node 5

Contains vertices $(1, 3, [1, 1])$ and $(2, 2, [1, 1])$.

### 3.3.2 The Stick Figure

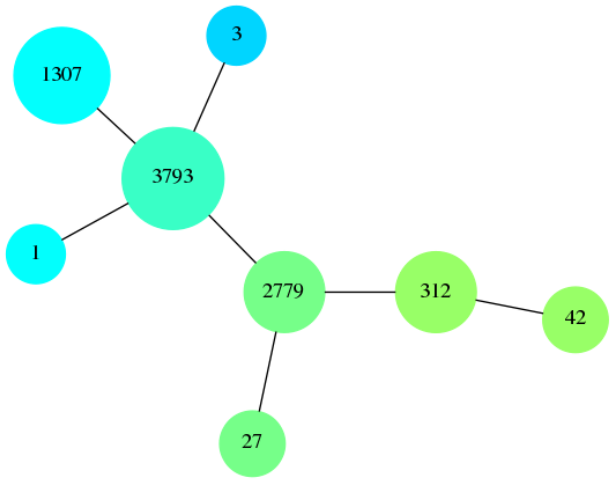



**Figure 16**: The Stick Figure feature from the `Mapper` output

Figure 16 is the feature we have named the Stick Figure. The clusters are named for their apparent location on the "stick figure" (left and right refer to the reader's left and right) and if needed, numbered from left to right.

Head Node

Contains vertices $(1, 1, [1, 2])$, $(1, 2, [1, 2])$, and $(1, 3, [0, 2])$.

Torso Node

Contains vertices $(1, 1, [1, 2])$, $(1, 2, [1, 2])$, $(1, 3, [0, 2])$, $(1, 3, [1, 2])$, $(1, 4, [0, 2])$, $(2, 1, [1, 2])$, $(2, 2, [1, 2])$, $(2, 3, [0, 2])$, and $(2, 3, [1, 2])$.

Left Arm Node

Contains the vertex $(2, 1, [1, 2])$.

Right Arm Node

Contains the vertex $(2, 3, [0, 2])$.

Pelvis Node

Contains vertices $(1, 2, [2, 2])$, $(1, 3, [1, 2])$, $(1, 4, [0, 2])$, $(1, 4[1, 2])$, $(2, 2, [1, 2])$, and $(2, 3, [1, 2])$.

Left Leg Node

Contains the vertex $(1, 2, [2, 2])$.

Right Leg Node 1

Contains the vertices $(1, 3, [2, 2])$ and $(1, 4, [1, 2])$.

Right Leg Node 2

Contains the vertex $(1, 3, [2, 2])$.

### 3.3.3 The Flower

Figure 17 is the feature we have named the Flower. The clusters are named either "Petals" for the group of clusters containing only one graph on the left-hand side of the figure, or "Stem 1" through "Stem 5" moving from left to right along the remaining clusters.

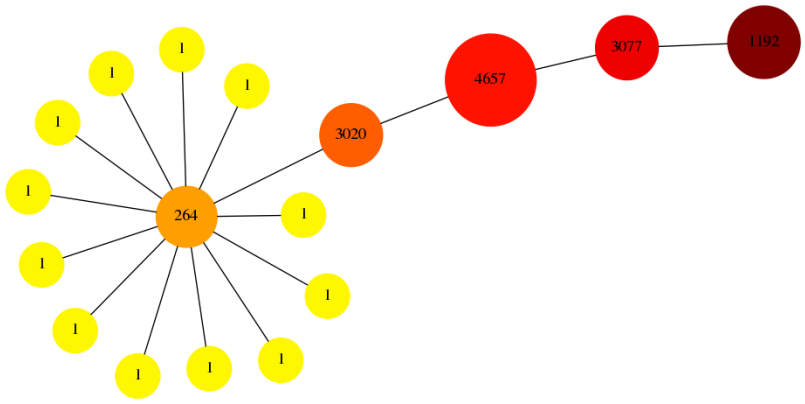



**Figure 17**: The Flower feature from the `Mapper` output

Petal Nodes

All 12 of the petals are the same vertex — $(1, 3, [0, 3])$ — they appear separately because of an odd artifact induced by the histogram cutoff.

Stem Node 1

Contains the vertices $(1, 1, [1, 3])$, $(1, 2, [1, 3])$, and $(1, 3, [0, 3])$.

Stem Node 2

Contains the vertices $(1, 1, [1, 3])$, $(1, 2, [1, 3])$, $(1, 3, [1, 3])$, and $(1, 4, [1, 3])$.

Stem Node 3

Contains the vertices $(1, 2, [2, 3])$, $(1, 3, [1, 3])$, $(1, 3, [2, 3])$, and $(1, 4, [1, 3])$.

Stem Node 4

Contains the vertices $(1, 2, [2, 3])$, $(1, 3, [2, 3])$, and $(1, 4, [2, 3])$.

Stem Node r5

Contains the vertices $(1, 4, [2, 3])$ and $(1, 4, [3, 3])$.

### 3.3.4 The noise

Figure 18 is a collection of several "noisy" nodes. They are numbered from top to bottom (starting at the pair labeled 5).

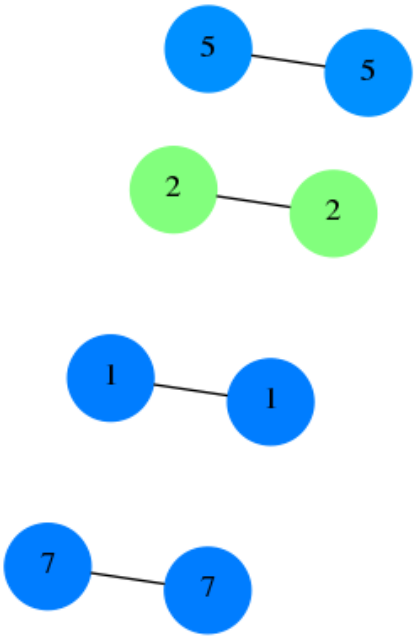



**Figure 18**: The noise nodes from the `Mapper` output

Cluster 1

Contains the vertex $(1, 2, [0, 2])$.

Cluster 2

Contains the vertex $(2, 2, [2, 2])$.

Cluster 3

Contains the vertex $(1, 1, [0, 2])$.

Cluster 4

Contains the vertex $(2, 3, [1, 1])$.

# Chapter 4
# Conclusion and Further Questions

## 4.1 Conclusions

Through our analysis there appeared three stable features plus several nodes that appear to be noise from the clustering cut-off method, outlined in Section 3.3. We address some observations of each of these features, as well as our conclusions of what these features tell us about the underlying assembly graphs and their invariants.

All graphs of DOWs were generated by a program written jointly by Jonthan Burns and Egor Dolzhenko which is available online at `http://knot.math.usf.edu`

### 4.1.1 The Line

The line formation tends to contain the simplest graphs, with relatively low nesting indices, and mostly low assembly number graphs. There is not a graph with a genus higher than 2 in the whole feature. This feature contains the least number of data points of the three main features.

### 4.1.2 The Stick Figure

The graphs in stick figure formation have the second highest genera in general, and tend to contain graphs with slightly higher assembly numbers and nesting indices. An interesting feature is that this graph has the most complex graphs with assembly number 2 (at least with this data set). This feature contains the second most data points of the three main features, one node containing more than all of the data points in the line feature.

### 4.1.3 The Flower

The graphs in the flower are some of the most complex, but they are all assembly number 1. This might suggest that there is only so much structural complexity that can be found in graphs of the size we are

considering, and some sort of cumulative effect between assembly numbering and the genus range of a graph. This feature also contains the most data points of the three main features.

An interesting feature of the flower is the spread of 12 "petals" that appears on the lefthand side of the formation. This peculiar formation occurred because the only graphs that were mapped into this particular partition set had an assembly number of 1, a nesting index of 3, and a genus range of 0 to 3; in other words all 12 graphs were represented by the same point. When the algorithm began to try to cluster this partition set, it found no nearby clusters (as the algorithm requires positive distance between clusters to work). Thus the algorithm returned each data point in its own cluster. Ideally, these all would be considered together as a single point. We expect that modifying the clustering algorithm to assume a minimum clustering distance of zero would eliminate this odd feature.

### 4.1.4 The Noise

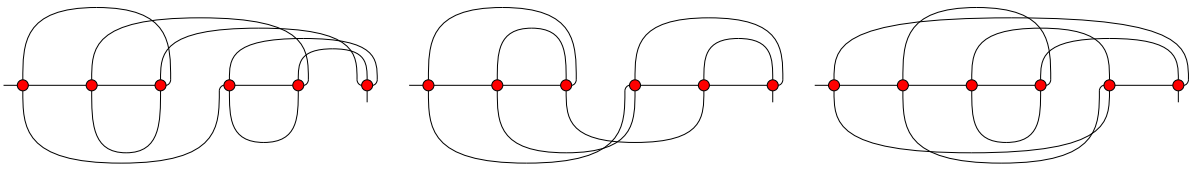
It is not yet clear whether the noise nodes have any special significance or are simply graphs that weren't clustered properly. They tend to be very small groups that fall in between other large clusters.

### 4.1.5 Analysis

The existence of clusters with all spreads of relations between assembly number and nesting indices lends further weight to Arredondo's work in [3]. But the fact remains that the majority of the graphs fall in clusters with close relations between the two lend some weight to there being a combinatorial connection between the two, given that the same pattern appears in a similar analysis of the biological data from [21].

The strongest link amongst nodes and inside features though can clearly be seen when we look at a coloring of the original data as in Figure 24. We find that the strongest correlation is the genera of the graphs, which forms strata in the graph. The delineation between different features corresponds perfectly with delineation between genus ranges.

There also seems to be a weak, inverse relationship between the maximum genus of a graph and the assembly number, given by the absence of graphs with assembly number 2 and a maximum genus of 3. Furthermore there does seem to be some correlation between a graph's nesting index and the genera it attains, though the connection is far from concrete. This is perhaps an area for further investigation.



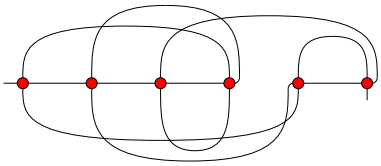
(a) Graph of 123145236456



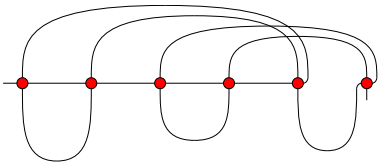
(b) Graph of 123145642356



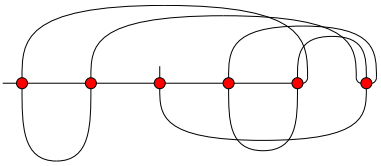
(c) Graph of 123425615346



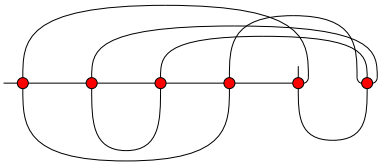
(d) Graph of 123425634156



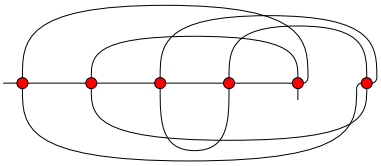
(e) Graph of 123451256346



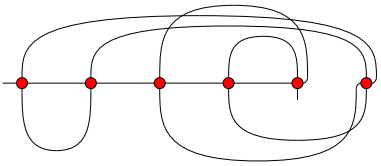
(f) Graph of 123451264563



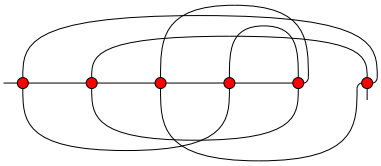
(g) Graph of 123451462365



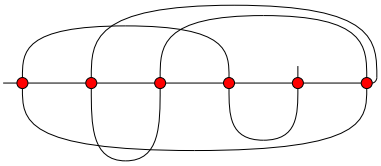
(h) Graph of 123451634625



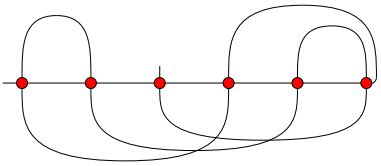
(i) Graph of 123453612645



(j) Graph of 123453614526

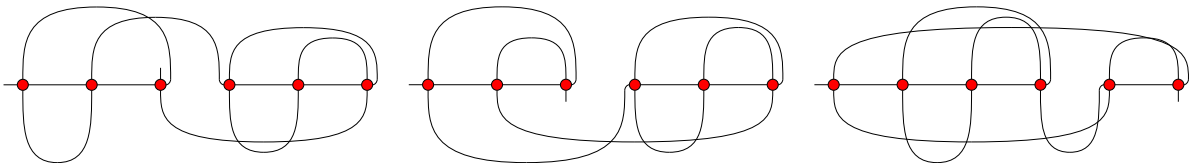


(k) Graph of 123456236145



(l) Graph of 123456412563

**Figure 19**: The assembly words and graphs found in the Petals nodes with assembly number 1, nesting index 3, and genus range of 0 to 3



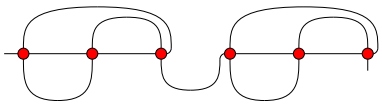
(a) Graph of 123124564563 (b) Graph of 123145645623 (c) Graph of 123423456156

(d) Graph of 123453456126 (e) Graph of 123456456123

**Figure 20**: The assembly words and graphs found in the node Noise 1 with assembly number 1, nesting index 2, and genus range of 0 to 2

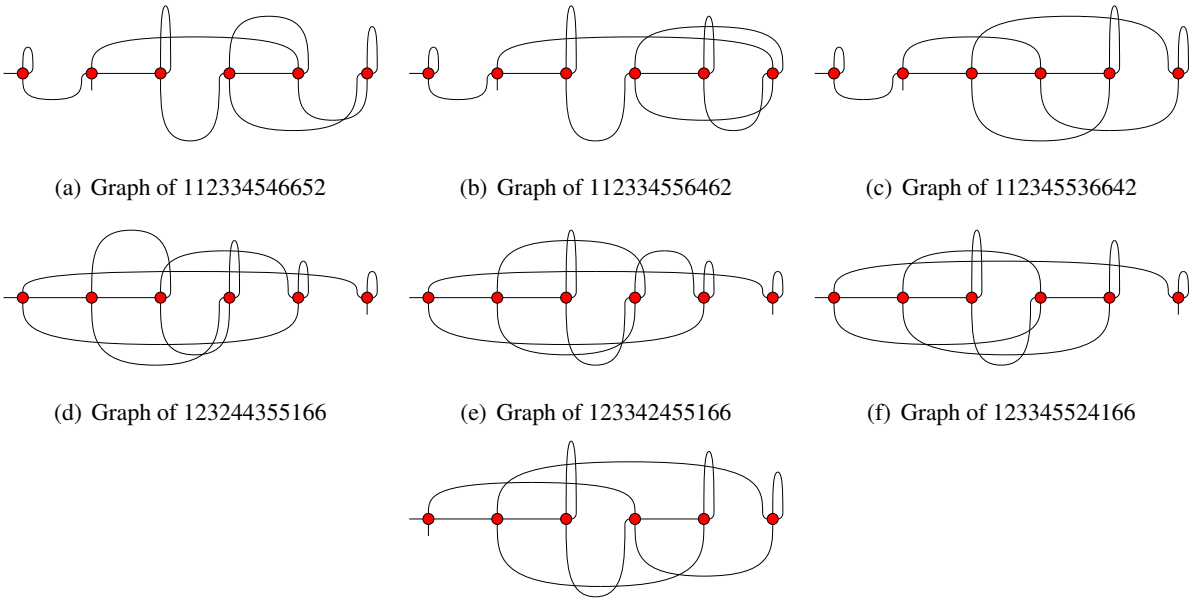
(a) Graph of 112343425656 (b) Graph of 121234545366

**Figure 21**: The assembly words and graphs found in the node Noise 2 with assembly number 2, nesting index 2, and a genus of 2



(a) Graph of 123123456456

**Figure 22**: The assembly words and graphs found in the node Noise 3 with assembly number 1, nesting index 1, and genus range of 0 to 2



(a) Graph of 112334546652 (b) Graph of 112334556462 (c) Graph of 112345536642

(d) Graph of 123244355166 (e) Graph of 123342455166 (f) Graph of 123345524166

(g) Graph of 123345526641

**Figure 23**: The assembly words and graphs found in the node Noise 4 with assembly number 2, nesting index 3, and genus of 1

### 4.2 Further Questions

There is much room to expand this dataset used in the computations presented here. Due to computation limitations, we were forced to consider only assembly graphs up to 6 vertices, while the data for up to 7 vertex assembly graphs is readily available and further data can be generated quite easily (the only cost being time). However such computations would take quite a long time — the pairwise distance matrix used in this paper was only a 165 MB *.mat file, but the pairwise distance matrix for assembly graphs of up to 7 vertices is a 361.2 **GB** *.mat file. If this challenge can be addressed by parallelizing the code for `Mapper`, or writing a version of the algorithm that runs in a more efficient language, the additional data in these graphs could greatly expand the value of this analysis.

There seems to be some suggestion of a relationship between the nesting index of a graph and its genus range. Though this does not seem to be a strong correlation it is worth investigating further, as does a possible inverse relation between the assembly number of a graph and its maximum genus.

Another question that we have uncovered is whether the assembly graphs generated by biological data from [21] will closely resemble the results we have found here or whether they will appear quite different

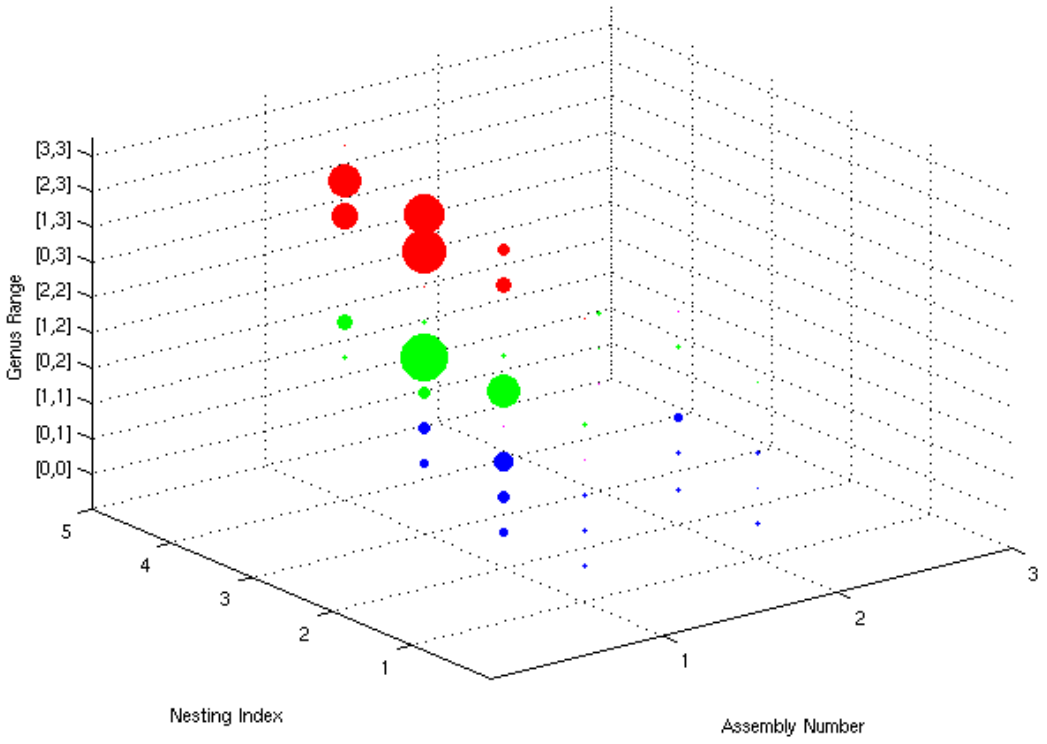



**Figure 24**: The original data, colored to indicate which cluster it will be mapped to. Blue stands for line clusters, green for stick figure clusters, red for flower clusters, and yellow for noise clusters

— perhaps only containing a subset of the features we have found here. It is our intuition that the second scenario is likely the case, and that evolutionary complexity of a recombination as measured by the nesting index will closely correlate to the mathematical complexity of the resultant assembly graph as measured by the genus range when we examine assembly graphs that correspond to recombination of *O. trifallax*.

## References


[1] Angeleska, A., N. Jonoska, and M. Saito, 2009: DNA Recombination through assembly graphs, *Discrete Appl. Math.*, **157**, pp. 3020-3037. doi: 10.1016/j.dam.2009.06.011.

[2] Arredondo, R., 2013: Reductions on Double Occurrence Words, *Congr. Numer.*, **218**, pp. 43-56.

[3] Arredondo, R., 2014: *Properties of Graphs Used to Model DNA Recombination*. Master's Thesis, Department of Mathematics and Statistics, University of South Florida.

[4] Bracht, J.R., W. Fang, A.D. Goldman, E. Dolzhenko, et. al, 2013: Genomes on the Edge: Programmed Genome Instability in Ciliates, *Cell*, **152**, pp. 406-416. doi: 10.1016/j.cell.2013.01.005.

[5] Buck, D., E. Dolzhenko, N. Jonoska, M. Saito, and K. Valencia, 2013: Genus Ranges of 4-Regular Rigid Vertex Graphs, Submitted.

[6] Burns, J., E. Dolzhenko, N. Jonoska, T. Muche, and M. Saito, 2013: Four-regular graphs with rigid vertices associated to DNA recombination, *Discrete Appl. Math.*, doi:10.1016/j.dam.2013.003.

[7] Carlsson, G., 2009: Topology and data, *Bull. Amer. Math. Soc.*, **46**, pp. 225-308.

[8] DuBois M,, D.M. Prescott, 1995: Scrambling of the actin I gene in two Oxytricha species. *Proc. Natl. Acad. Sci. USA*, **92**, pp. 3888-3892.

[9] Kauffman, L. H., 1989: Invariants of Graphs in Three-Space, *Trans. Amer. Math. Soc.*, **311**, pp. 697-710.

[10] Lafon, S. and A.B. Lee Diffusion Maps and Coarse-Graining:A Unified Framework for Dimensionality Reduction, Graph Partitioning, and Data Set Parameterization, 2006: *IEEE Trans. Pattern Anal. Mach. Intell.*, **28**, pp. 1393-1403.

[11] Lum, P.Y., G. Singh, A. Lehman, T. Ishjanov, M. Vejdemo-Johansson, et. al, 2013: Extracting insights from the shape of complex data using topology, *Sci. Rep.*, **3**, 1236. doi: 10.1038/srep01236.

[12] MATLAB Release 2013b, The MathWorks, Inc., Natick, Massachusetts, United States.

[13] Müllner, D. and A. Babu, 2013: *Python Mapper: An open-source toolchain for data exploration, analysis, and visualization*, URL `http://math.stanford.edu/ muellner/mapper`.

[14] Munkres, J.R., 2000: *Topology*. 2nd ed. Pearson, 527 pp.

[15] Munkres, J.R., 1984: *Elements of Algebraic Topology*. Perseus Books, 454 pp.

[16] Nanda, V. and R. Sazdanović, 2014: Simplicial Models and Topological Inference in Biological Systems. *Discrete and Topological Models Molecular Biology*, Nataša Jonoska and Masahico Saito, Eds., Springer-Verlag, pp. 109-141.

[17] Partee, B., A. ter Meulen, and R. Wall, 1990: *Mathematical Methods in Linguistics*. Keuler, 669 pp.

[18] Prescott, D.M., 2000: Genome gymnastics: unique modes of dna evolution and processing in ciliates. *Nat. Rev. Genet.*, **1**, pp. 191-198. doi:10.1038/35042057.

[19] Silverman, B.W., 1986: Density Estimation for Statistics and Data Analysis. *Monogr. Statist. Appl. Probab.*, **26**, Chapman and Hall, London.

[20] Singh, G, F. Mémoli, and G. Carlsson, 2007: Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition. *Proc. Eurographics Symp. on Point-Based Graphics*, Czech Technical University, Prauge, The Eurographics Assoc., pp. 91-100.

[21] Swart E.C., J.R. Bracht, V. Magrini, P. Minx , X. Chen, et al. (2013): The Oxytricha trifallax Macronuclear Genome: A Complex Eukaryotic Genome with 16,000 Tiny Chromosomes. *PLoS Biol.*, **11**, e1001473. doi:10.1371/journal.pbio.1001473.

## Appendix

## Additional Mapper Outputs

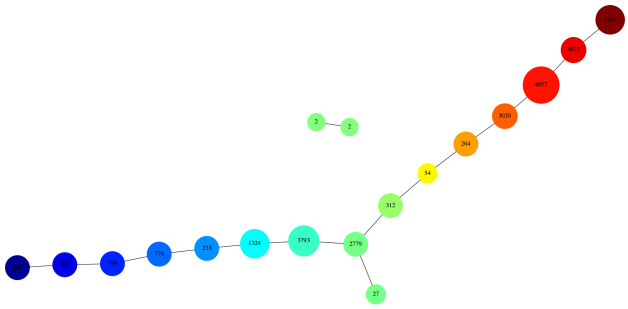


**Figure 25**: Output of `Mapper` with histogram cutoff $h$ set to 12, partition resolution of $r = \frac{1}{8}$, and overlap parameter of $p = .5$

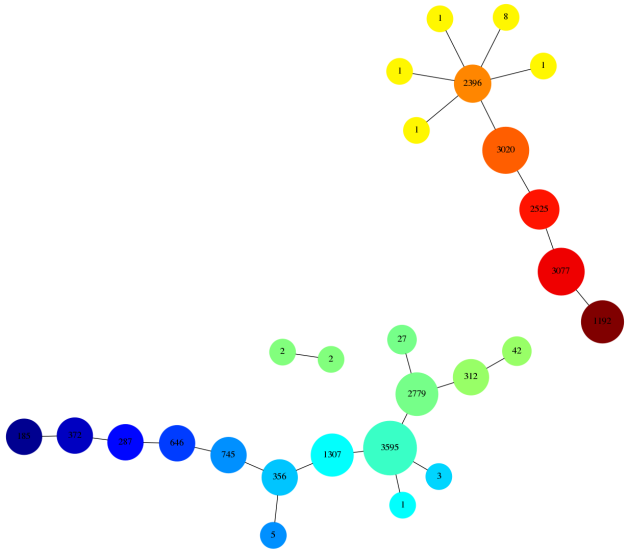



**Figure 26**: Output of `Mapper` with histogram cutoff $h$ set to 12, partition resolution of $r = \frac{1}{9}$, and overlap parameter of $p = .5$

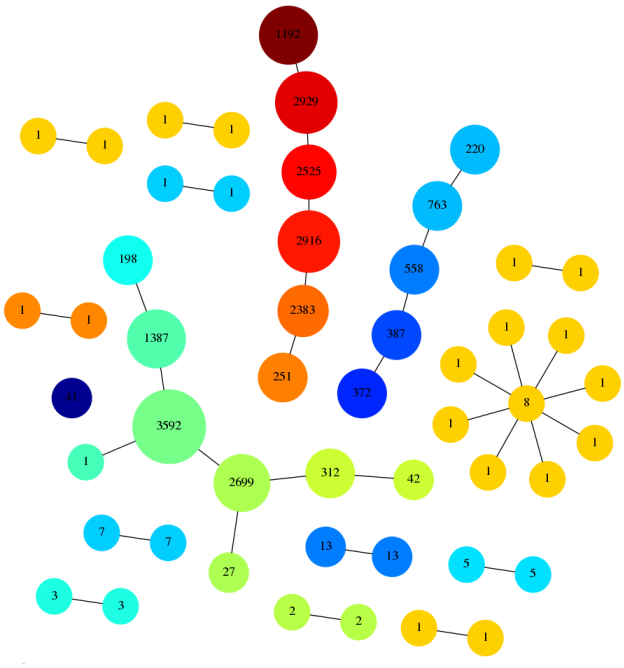



**Figure 27**: Output of `Mapper` with histogram cutoff $h$ set to 15, partition resolution of $p = \frac{1}{8}$, and overlap parameter of $p = .5$