Thesis Proposal

Visualization methods for genealogical and RNA-sequencing datasets

Lindsay Rutter

Program of Study Committee:

Dianne Cook (Major Professor)
Amy Toth (Major Professor)
Heike Hofmann
Daniel Nettleton
James Reecy

May 16, 2016

Contents

Chapter 1

Introduction

1.1 Background to data visualization

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.2 Problems to be addressed

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

1.3 Overview of thesis research

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis,

molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Chapter 2

ggenealogy: Visualization methods for genealogical datasets

2.1 Abstract

In this chapter, we introduce `ggenealogy` (Rutter et al. 2015), a developing `R` software package that provides tools for searching through genealogical data, generating basic statistics on their graphical structures using parent and child connections, and displaying the results. The package allows users to draw the genealogy in relation to variables related to the nodes, and to determine and display the shortest path distances between the nodes. Production of pairwise distance matrices and genealogical diagrams constrained on generation are also available in the visualization toolkit. We have tested the tools on a dataset with milestone cultivars of soybean varieties (Hymowitz et al. 1977) as well as on a web-based database of the academic genealogy of mathematicians (North Dakota State University and American Mathematical Society).

The software package has been available on the Comprehensive `R` Archive Network since March 2015 (Rutter et al. 2015). In August 2015, a paper introducing the `ggenealogy` software package received a student paper competition award from the Statistical Computing and Graphics Section of the American Statistical Association. Currently, we are preparing to submit another paper describing the software package to the Journal of Statistical Software.

2.2 Introduction

Genealogy is the study of parent-child relationships. By tracing through parent-child lineages, genealogists can study the histories of features that have been modified over time. Comparative geneticists, computational biologists, and bioinformaticians commonly use

genealogical tools to better understand the histories of novel traits arising across biological lineages. For example, desirable modifications in crops could include an increase in protein yield or an increase in disease resistance, and genealogical structures could be used to assess how these desirable traits developed. At the same time, genealogical lineages can also be used to assess detrimental features, such as to determine the origin of hazardous traits in rapidly-evolving viruses.

Genealogical structures can also serve as informative tools outside of a strict biological sense. For instance, we can trace mentoring relationships between students and dissertation supervisors with the use of academic genealogies. This can allow us to understand the position of one member in the larger historical picture of academia, and to accurately preserve past relationships for the knowledge of future generations. Similarly, linguistic genealogies can be used to decipher the historical changes of vocabulary and grammatical features across related languages. In short, there is a diverse array of disciplines that can elicit useful information about features of interest by using genealogical data.

In all these examples, the genealogical relationships can be represented visually. Access to various types of plotting tools can allow scientists and others to more efficiently and accurately explore features of interest across the genealogy. We introduce here a developing visualization toolkit that is intended to assist users in their exploration and analysis of genealogical structures. In this paper, we demonstrate the main tools of the software package `ggenealogy` using two example genealogical datasets, one of soybean cultivars (Hymowitz et al. 1977) and the other of academic mathematicians (North Dakota State University and American Mathematical Society).

2.3 Available Software

Publishing in the open source `R` statistical programming language allows for tools to be distributed and modified at ease, encourages cross-platform collaboration, and provides a foundation for effective and aesthetic data visualization from the grammar of graphics. There are several useful `R` packages that offer tools for analyzing and visualizing genealogical datasets. Here, we introduce these packages, and emphasize the new features that `ggenealogy` brings to this collection of work.

The `R` package `pedigree` is named after the standardized chart used to study human family lines, and sometimes used to select breeding of animals, such as show dogs (Coster 2013). This package does provide tools that perform methods on parent-child datasets, such as rapidly determining the generation count for each member in the pedigree. However, it does not provide any visualization tools.

Another `R` package called `kinship2` does produce basic pedigree charts (Therneau et al.

2015). In Figure 2.1, we provide an example pedigree chart from the `kinship2` package vignette. This pedigree chart adheres to the standard set of symbols used for visualizing genealogical structures: Males are represented with squares and females with circles. Parents are connected to each other by horizontal lines, and to their children by vertical lines. Siblings are connected by horizontal sibship lines. Even though this standard pedigree chart creates powerful charts that can be applied across many applications, it cannot provide unequivocal information in situations where inter-generational breeding occurs, as is often the case in agronomic genealogical lineages.

We demonstrate how the standardized pedigree charts in the `kinship2` package generate ambiguous results in such scenarios by superimposing a hypothetical inter-generational breeding case in Figure 2.1. In that figure, each generation is defined by its position on the vertical axis, with the first generation containing individuals 201 and 202. We superimposed green-highlighted individual 215 onto the pedigree chart for explanatory purposes. Its parents are individuals 201 and 206, which are from generations one and two, respectively, and have a parent-child relationship between themselves. As an offspring of a parent-child relationship, individual 215 is both a second and third generation individual. Hence, individual 215 should be displayed in both second and third generational positions on the vertical axis. However, standard pedigree tools only allow for an individual to be displayed once. As a result, in special cases where inter-generational breading occurs, such as in agronomic applications, standardized tools for visualizing genealogical information ambiguously portray the genealogical dataset.

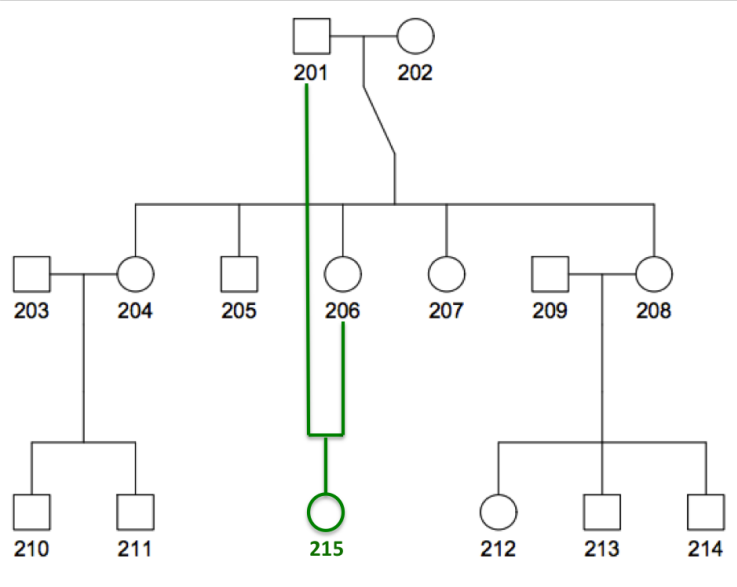

Figure 2.1: Example pedigree chart from the `kinship2` package, where the vertical axis denotes generation count. We superimposed green-highlighted individual 215 for explanatory purposes. As an offspring of a parent-child relationship, individual 215 is both a second and third generation individual. Hence, it should be displayed twice on the vertical axis, once for each of its generation counts. However, standard pedigree tools only allow for an individual to be displayed once, leading to ambiguous portrayal of the genealogical dataset.

In addition, popular graph drawing software such as `GraphViz` and `Cytoscape` can be used to visualize genealogical structures (Gansner and North 2000, Shannon et al. 2003). Graphs are defined as objects with sets of nodes and edges, where sets indicate that their comprised elements cannot be repeated. In other words, graphical structures do not allow for repeated nodes, and hence, as is the case with the aforementioned `R` packages, these popular graph plotting software cannot precisely portray the genealogical dataset in cases of inter-generational breeding.

We again illustrate this problem in Figure 2.2 with an example genealogy using popular graph drawing software like `GraphViz` and `Cytoscape`. Here, generation count is denoted by the vertical axis. As was shown in Figure 2.1, here too we superimpose a green node that has parents from two different generations. This green node is both a second and third generation individual, and should be displayed in both corresponding generation positions on the vertical axis. However, standard graph visualization tools only allow for a given node to be displayed once. As a result, this green node must be ambiguously positioned in either the second or third generation position; in the figure, it is denoted as a third generation individual. In Section 2.9, we will demonstrate `ggenealogy` plots that can remedy these problems.

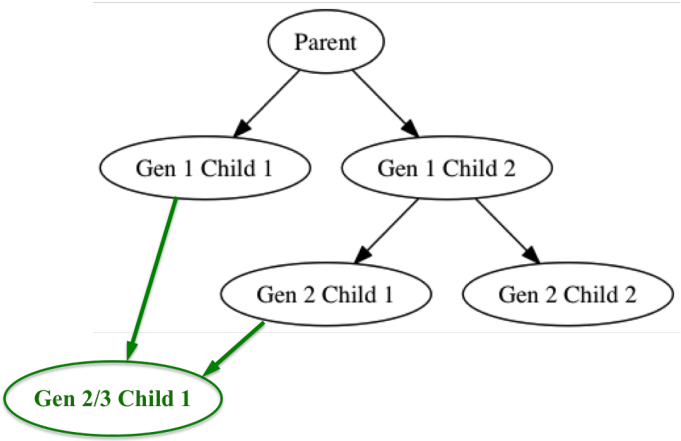

Figure 2.2: Example genealogical display using popular graph software like `GraphViz` and `Cytoscape`, with generation count denoted by the vertical axis. As was shown in Figure 2.1, the green node has parents from two different generations, and hence must be ambiguously positioned as one of two generation counts.

2.4 Example Datasets

The `ggenealogy` package comes with two example datasets, one comprises a soybean genealogy and the other comprises an academic statistician genealogy. We will introduce both example datasets to demonstrate some of the tools available in the software.

2.4.1 Soybean Genealogy

We start with the soybean genealogy, which is available as a data frame structure with 390 rows and five columns. These data were collected from field trials, genetic studies, and United States Department of Agriculture (USDA) bulletins, and date as early as the first decade of the 1900s. They contain information on the copy number variants, single nucleotide polymorphisms, and protein content for each of the varieties, although we removed that information for a succinct example dataset. In this context, the software could ideally be used by agronomists who wish to study how soybean varieties are related. By referencing the visualization of the genealogical structure, these scientists may better understand genetic testing results - in this particular dataset, in terms of copy number variants, single nucleotide polymorphisms, protein content, and yield - and use that knowledge in future breeding decisions.

Each row contains information about a particular child soybean variety, including the name of the child, its yield, the year it was released, whether or not its release year was imputed, and the name of its parent. It should be noted that it typically requires many crosses over the span of one to two decades to develop a new variety that has introduced a desired trait and/or removed an undesired trait. Hence, the release year variable in this dataset represents the year in which the variety was released to the public after its development

period. While the name of the child is required, the other four columns can have missing values (which are represented in R with the symbol NA for "not available"). As a result, while each row does contain information about a particular child soybean variety, whether or not a given row also contains information about a parent-child relationship between a pair of soybeans depends on whether or not the parent column has a missing value.

In total, there are 230 soybean varieties in the dataset, 206 of which are children and 165 of which are parents. There are soybeans that are both children and parents. Of the children, 156 have two parents, 28 have one parent, and 22 have zero parents. There are 340 parent-child relationships in the dataset.

We can load the example dataset of soybean genealogy (sbGeneal) and examine its structure.

```
R> install.packages("ggenealogy")
R> library("ggenealogy")
R> data(sbGeneal)
R> str(sbGeneal)
```

```
'data.frame':  390 obs.  of 5 variables:
$ child :  chr "5601T" "Adams" "A.K." "A.K. (Harrow)" ...
$ year :  num 1981 1948 1910 1912 1968 ...
$ yield :  int NA 2734 NA 2665 NA 2981 2887 2817 NA NA ...
$ year.imputed:  logi TRUE FALSE TRUE FALSE FALSE FALSE ...
$ parent :  chr "Hutcheson" "Dunfield" NA "A.K." ...
```

2.4.2 Academic Genealogy of Statisticians

The ggenealogy package also comes with an academic genealogy of statisticians; this dataset is in the form of a data frame with 8165 rows and six columns. To develop this later dataset, we contacted the Math Genealogy Project (North Dakota State University and American Mathematical Society), a web-based database for the genealogy of academic mathematicians. This database, which currently contains almost 200,000 entries, is a service of the North Dakota State University Department of Mathematics and the American Mathematical Society. The Mathematics Genealogy Project contact provided us a Structured Query Language (SQL) export, and we used PostgreSQL to query the database (PostgreSQL).

Each entry in the database contained 26 variables pertaining to an individual who received a graduate-level academic degree in mathematics. One of these variables was called "msc" (Mathematics Subject Classification), and we selected only those entries that contained

a value of 62 for this variable (coded as "Statistics"). Furthermore, we only retained entries that had a parent if that parent was also in the field of "Statistics". Hence, in our parent-child relationships, both the child and the parent received postbaccalaureate degrees in statistics, and the parent was the academic advisor to the child. This process resulted in 8995 entries, which we reduced to 8165 entries by removing duplicate entries. With the final data frame of 8165 entries, we only maintained six of the original 26 variables.

Each row of the final data frame contains information about a particular child who received a graduate-level academic degree in statistics, including the name of the child, the year the child obtained the degree, the country and school from which the child obtained the degree, the thesis title of the degree awarded to the child, and the name of its parent. There are no missing values for the country and school from which the child received its degree or the name of the child; however, some of the years contain missing values (NA), and some of the parent and thesis names contain empty strings (""). As a result, while each row does contain information about a particular child, whether or not a row also contains information about a parent-child relationship between a pair of academic statisticians depends on whether or not the parent column has an empty string.

In total, there are 7122 individuals in the dataset, 7122 of which are children and 872 of which are parents. Every parent is also a child, but not every child is also a parent. Of the children, two have four parents, ten have three parents, 226 have two parents, 2801 have one parent, and 4083 have no parents. There are 3291 parent-child relationships in the dataset.

We can load the example dataset of academic genealogy of statisticians (`statGeneal`) and examine its structure.

```
R> data(statGeneal)
R> dim(statGeneal)
```

```
[1] 8165 6
```

```
R> colnames(statGeneal)
```

```
[1] "child" "parent" "year" "country" "school" "thesis"
```

2.5 Genealogical Input Format

As is the case with both example data files introduced above, `ggenealogy` requires that the genealogy input file is a data frame structure with at least two columns. One column

must be labeled "child", and each case in that column must be of type character. The other column must be labeled "parent," and each case in that column must either be of type character, type NA, or type "". At this point, any ggenealogy plot that only requires information about parent-child relationships can be used.

However, some ggenealogy plots also make use of quantitative variable values associated with individuals in the genealogy. For these plots, the input data frame should also contain a third column. In both example data files, this column is labeled "year," and each case in that column can either be of type numeric, type NA, or type "". At this point, any ggenealogy plot can be used.

2.6 Generating a Graphical Object

Most functions in the ggenealogy software package require an input parameter of a graph structure. Therefore, as a preprocessing step, we must first convert our original data frame structure into a graph structure. Below, we read in the R data file sbGeneal that is included in the package as a sample data set of soybean genealogy.

We now convert it into an igraph object (Csardi and Nepusz 2006) sbIG using the function dfToIG().

```
R> sbIG <- dfToIG(sbGeneal)
R> sbIG
```

```
IGRAPH UNW- 230 340 -
+ attr:  name (v/c), weight (e/n)
+ edges (vertex names):
[1] 5601T -Hutcheson Adams -Dunfield
[3] A.K. -A.K. (Harrow) Altona -Flambeau
[5] Amcor -Amsoy 71 Adams -Amsoy
[7] Amsoy 71 -C1253 Anderson -Lincoln
[9] Bay -York Bedford -Forrest
[11] Beeson -Kent Blackhawk-Richland
[13] Bonus -C1266R Bradley -J74-39
[15] Bragg -Jackson Bragg -Bragg x D60-7965
+ ...  omitted several edges
```

There are many statistics about the sbGeneal dataset that we may wish to know that cannot easily be obtained through images and tables. The package function getBasicStatistics() can be called, using the sbIG object as input. This will return a list of common graph theoretical measurements regarding the genealogical structure. For

instance, is the whole structure connected? If not, how many separated components does it contain? In addition to these statistics, the `getBasicStatistics()` function will also return the number of nodes, the number of edges, the average path length, the graph diameter, and other graph theoretical information.

```
R> getBasicStatistics(sbIG)
```

```
$isConnected
[1] FALSE

$numComponents
[1] 11

$avePathLength
[1] 5.333746

$graphDiameter
[1] 13

$numNodes
[1] 230

$numEdges
[1] 340

$logN
[1] 5.438079
```

2.7 Plotting a Shortest Path

With soybean lineages, it may be useful for soybean breeders to track how two varieties are related to each other via parent-child relationships. Then, any dramatic changes in yield and other measures of interest between the two varieties can be traced across their genetic timeline. The `ggenealogy` package allows users to select two varieties of interest, and determine the shortest pathway of parent-child relationships between them, using the `getPath()` function. This will return a list that contains the variety names and their years in the path.

```
R> pathTN <- getPath("Tokyo", "Narow", sbIG, sbGeneal)
R> pathTN
```

```
$pathVertices
[1] "Tokyo" "Volstate" "Jackson" "R66-873" "Narow"

$yearVertices
[1] "1907" "1942" "1954.5" "1971.5" "1985"
```

The returned path object can then be plotted using the `plotPath()` function.

```
R> plotPath(pathTN)
```

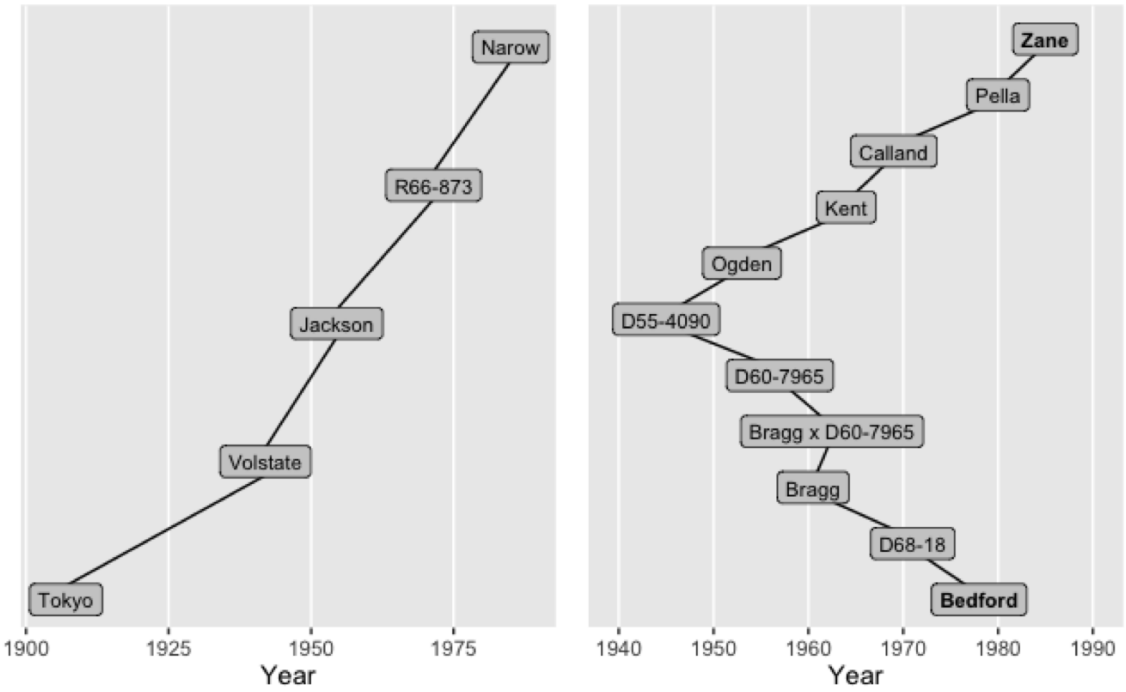

Figure 2.3: Left: The shortest path between varieties Tokyo and Narow is strictly composed of a unidirectional sequence of parent-child relationships. Right: The shortest path between varieties Zane and Bedford is not strictly composed of unidirectional parent-child relationships; they instead have a cousin-like relationship.

This produces a visual that informs users of all the varieties involved in the shortest path between the two varieties of interest (see left half of Figure 2.3). In this plot, the release year of all varieties involved in the path are indicated on the horizontal axis, while the vertical axis has no meaning other than to simply to display the labels evenly spaced vertically. The shortest path between varieties `Tokyo` and `Narow` is composed of a unidirectional

series of parent-child relationships, with `Tokyo` as the starting ancestor in the early 1900s, `Narow` as the most recent descendent in the mid 1980s, and three varieties in between.

Next, we can run the same set of functions on a different pair of varieties. First, a call to the `ggenealogy` function `getYear()` indicates that variety `Bedford` was released in 1978 and variety `Zane` in 1985.

```
R> getYear("Bedford", sbGeneal)
```

```
[1] 1978
```

```
R> getYear("Zane", sbGeneal)
```

```
[1] 1985
```

We can then create a plot showing the shortest path between these two varieties of interest. As this is a longer path, we may also consider setting the `fontFace` variable of the `plotPath()` to a value of 2, indicating we wish to boldface the two varieties of interest.

```
R> pathBZ <- getPath("Bedford", "Zane", sbIG, sbGeneal)
R> plotPath(pathBZ, fontFace = 2)
```

The resulting plot (right half of Figure 2.3) allows us to quickly determine that `Bedford` is not a parent, grandparent, or any great grandparent of `Zane`. Instead, we see that these two varieties are not related through a unidirectional parent-child lineage, but instead have a cousin-like relationship. The oldest common ancestor between `Zane` and `Bedford` is the variety `D55-4090`, which was released in the mid 1940s.

Furthermore, as determined by the figure, for both `Zane` and `Bedford`, there are four varieties of unidirectional parent-child relationships between each of them and their common ancestor `D55-4090`. Hence, any feature of interest that differentiates `Zane` and `Bedford` (protein content, yield, disease resistance, etc.) can also be examined across these two separate lineage histories.

2.8 Superimposing Shortest Path on Tree

Now that we can create path objects, we may wish to know how those paths are positioned compared to the entire genealogical lineage. For instance, of the documented soybean cultivar lineage varieties, where does the shortest path between two varieties of interest exist? Are these two varieties older compared to the overall data structure? Are they newer?

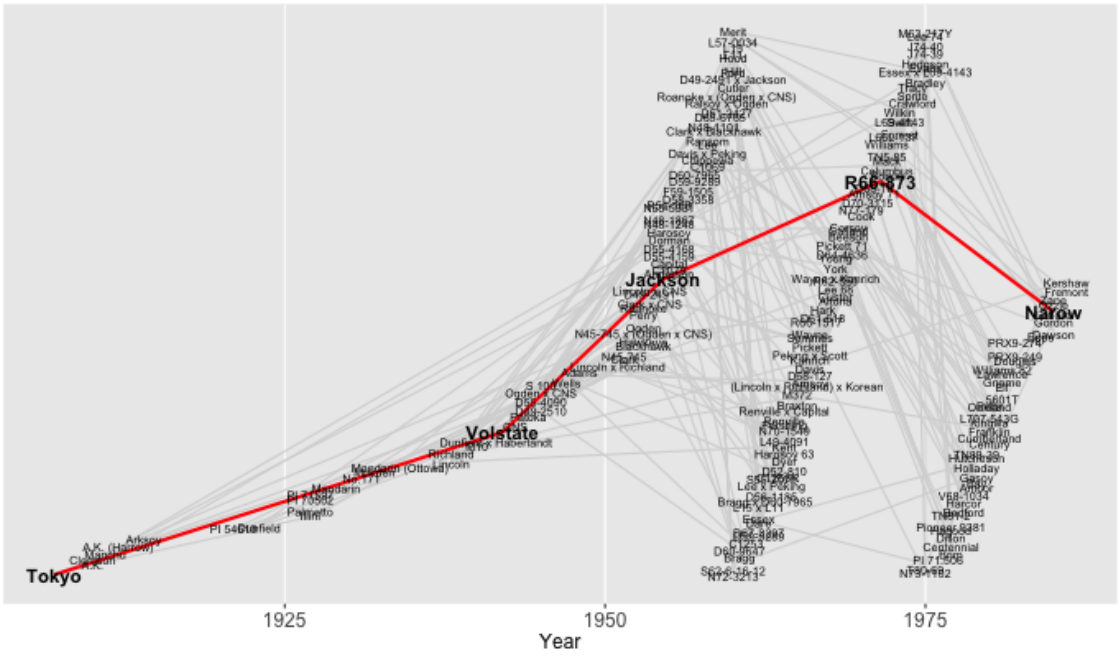

Figure 2.4: The shortest path between Tokyo and Narow, superimposed over the data structure, using a bin size of 3.

Or, do they span the entire structure, and represent two extreme ends of documented time points?

There is a function available in the `ggenealogy` package `plotPathOnAll()` that can allow users to quickly visualize their path of interest superimposed over all varieties and edges present in the whole data structure. Here we will produce a plot of the shortest path between varieties `Tokyo` and `Narow` across the entire dataset, as is displayed in Figure 2.4.

```
R> plotPathOnAll(pathTN, sbGeneal, sbIG, binVector = 1:3,
pathEdgeCol = "red", nodeSize = 2.5, pathNodeSize = 4) +
ggplot2::theme(axis.text = ggplot2::element_text(size = 12),
axis.title = ggplot2::element_text(size = 12))
```

In the code above, syntax from the `ggplot2` package was appended to the `plotPathOnAll()` function; this can be done for most `ggenealogy` functions (Wickham 2009). While the first three explicit parameters have been introduced earlier in this paper, the fourth parameter (`binVector`) requires some explanation. The motivation of the `plotPathOnAll()` function is to write node labels on a plot, with the center of each node label constricted on the horizontal axis to its quantitative variable of interest (in this case, year of release). As is the case for the plots before, the vertical axis has no meaning other than providing a plotting area in which to draw the node labels. Unfortunately, for large datasets, this motivation can be a difficult task because the text labels of the varieties

can overlap if they are assigned a similar y coordinate, have a similar year (x coordinate), and have long text labels (width of x coordinate).

For each variety, the x coordinate (year) and width of the x coordinate (text label width) cannot be altered, as they provide useful information. However, for each variety, the y coordinate is arbitrary. Hence, in an attempt to mitigate text overlapping, the `plotPathOnAll()` function does not randomly assign the y coordinate. Instead, it allows users to partially control the y coordinates with a user-determined number of bins (`binVector`).

If the user decides to produce a plot using three bins, as in the example code above, then the varieties are all grouped into three bins based on their year values. In other words, there will be bin 1 (the "oldest bin") which includes the one-third of varieties with the oldest years of release, bin 2 (the "middle bin"), and bin 3 (the "youngest bin"). Then, in order to decrease text overlap, the consecutively increasing y-axis coordinates are alternatively assigned to the three bins (For example: bin 1, bin 2, bin 3, bin 1, bin 2, bin 3, ...) repeatedly until all varieties are addressed. This algorithm means that for any pair of varieties within a given bin, there are exactly two other varieties vertically separating them.

In the code above, `binVector` was assigned a value of 3, and `pathEdgeCol` was assigned a value of "red". Additionally, we specified a size of 2.5 for the non-path node test using the `nodeSize` parameter, and a size of 4 for the path node text using the `pathNodeSize` parameter. There are several other parameters in the `plotPathOnAll()` function, which can be read in more detail using the help command.

This code resulted in Figure 2.4, where we see that edges not on the path of interest are thin and gray by default, whereas edges on the path of interest are bolded by default. We also see that variety labels in the path of interest are boldfaced by default. Figure 2.4 presents useful information: We immediately gather that the path of interest does span most of the years of the data structure. In fact, `Tokyo` appears to be the oldest variety in the dataset, and `Narow` appears to be one of the youngest varieties. We can also determine that the majority of varieties were released between 1950 and 1970.

However, Figure 2.4 has significant empty spaces between the noticeably distinct bins, whereas almost all text labels are overlapping, thereby decreasing their readability. To force text labels into these spaces, the user may consider using a larger number of bins. Hence, we next examine a bin size of 6 to create Figure 2.5.

```
R> plotPathOnAll(pathTN, sbGeneal, sbIG, binVector = 1:6,
pathEdgeCol = "seagreen2", nodeSize = 1, pathNodeSize = 3)
```

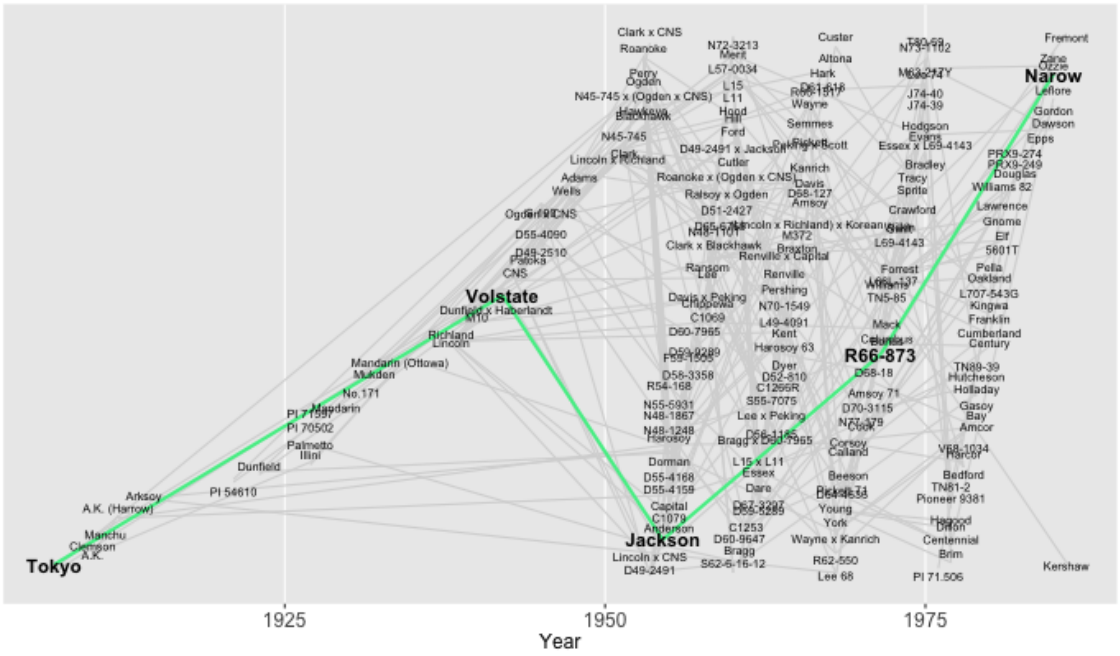

Figure 2.5: The shortest path between Tokyo and Narow, superimposed over the data structure, using a bin size of 6.

We can immediately see that Figure 2.5 more successfully mitigates text overlap compared to Figure 2.4. We can also confirm what we saw in the previous plot that indeed most varieties were released between 1950 and 1970, and any textual overlap is confined to this range of years.

2.9 Plotting Ancestors and Descendants by Generation

The most novel visual function in `ggenealogy`, `plotAncDes()` allows users to view the ancestors and descendants of a given variety. The inputted variety is highlighted in the center of the plot, ancestors are displayed to the left of the center, and descendants are displayed to the right of the center. The further from the center that a variety is located, the more generations that variety is distanced from the centered variety of interest.

This particular `ggenealogy` tool is unique because most available genealogy and graph visualization software do not allow for repeated labels. It is a useful tool because, as was demonstrated in Figures 2.1 and 2.2, some genealogical datasets require repeated node labels if they are to be visualized by generation counts. Indeed, our example soybean genealogy is one such dataset.

To demonstrate this tool, we will create a plot of the ancestors and descendants of the variety `Lee`. We specify that the maximum number of ancestor and descendant generations are both 6, and that the text of the variety of interest is highlighted in blue:

```
R> plotAncDes("Lee", sbGeneal, mAnc = 6, mDes = 6, vCol =
"blue")
```

This generates the top plot of Figure 2.6. We notice that `Lee` has 3 generations of ancestors and 5 generations of descendants. We also notice that some varieties are repeated in the plot, which is a unique feature provided by `ggenealogy`. For example, the variety `5601T` is represented four times - once as a third generation descendant of `Lee`, once as a fourth generation descendant of `Lee`, and twice as a fifth generation descendant of `Lee`. The variety `5601T` was repeated multiple times because there are multiple paths between `Lee` and `5601T`. For explanation purposes, all paths between `Lee` and `5601T` were manually highlighted in blue.

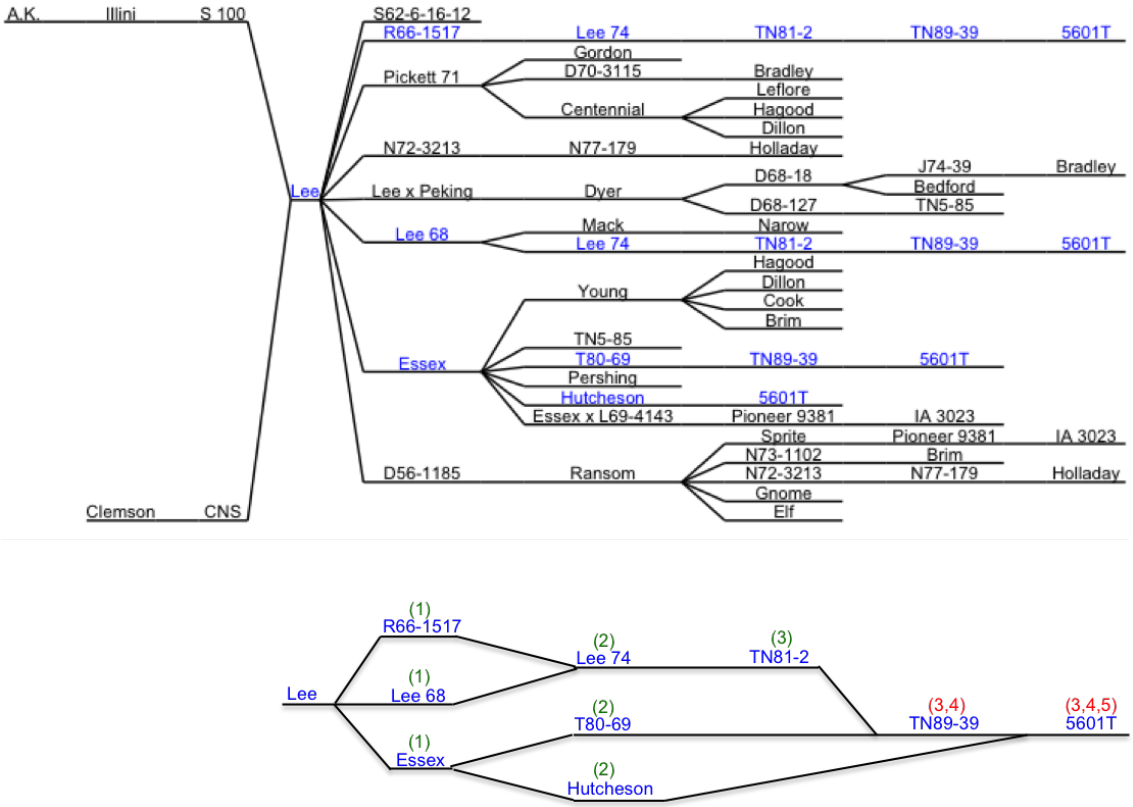

Figure 2.6: Top: All ancestors and descendants of the variety Lee are shown in this ggenealogy plot. Bottom: We now attempt to mimic the blue paths in the ggenealogy plot on the top, only now nodes cannot be repeated. The parenthetical numbers above each node represents the set of generation counts that node is away from the center node Lee. The presence of red parentheses indicate that the plot on the bottom ambiguously display the example soybean genealogy in the way that the ggenealogy plot on the top can accomplish.

The bottom plot of Figure 2.6 is not an output plot of `ggenealogy`. Instead, it was simply created for didactic purposes. Here, the paths that were manually highlighted in

blue in the top plot produced by ggenealogy are shown again, only now nodes cannot be repeated. The parenthetical number above each node represents the set of generation counts distancing that node from the center node Lee; green parentheses indicate that the node could be successfully placed in one horizontal position, but red parentheses indicate that the node could not be successfully placed in one horizontal position. We see that node TN89-39 cannot simultaneously be represented as both a third and fourth descendent of node Lee, and node 5601T cannot simultaneously be represented as a third, fourth, and fifth descendent of node Lee. Hence, without allowing nodes to repeat, this dataset cannot be presented in the graph on the bottom as it can be in the ggenealogy graph on the top. This is a current limitation in other genealogy and graphical software that ggenealogy can now provide.

2.10 Plotting Distance Matrix

It may also be of interest to generate matrices where the colors indicates a variable between all pairwise combinations of inputted varieties. The package ggenealogy also provides a function plotDegMatrix() for that purpose. We can demonstrate this function with the variable being the shortest path degree between a given pair of varieties. The shortest path degree is calculated as the smallest number of parent-child edges needed to traverse between two varieties of interest. For instance, in Figure 2.3, the shortest path degree between Tokyo and Narow is four and the shortest path degree between Bedford and Zane is ten.

Here we generate a distance matrix for a set of 10 varieties, setting the x-label and y-label as "Variety" and the legend label as "Degree". In this example, we add ggplot2 functionality to specify that pairs with small degrees are white, while those with large degrees are dark green, as well as to specify the text size of the legend title and label.

```
>R varieties <- c("Brim", "Bedford", "Calland", "Dillon",
"Hood", "Narow", "Pella", "Tokyo", "Young", "Zane")
>R plotDegMatrix(varieties, sbIG, sbGeneal, "Variety",
"Variety", "Degree") + ggplot2::scale_fill_continuous(low =
"white", high = "darkgreen") + ggplot2::theme(legend.title
= ggplot2::element_text(size = 15), legend.text =
ggplot2::element_text(size = 15))
```

This creates the plot in Figure 2.7. We see that the degree of the shortest path between varieties Bedford and Zane is 10, which is consistent with what we saw earlier in Figure 2.3. However, we now also see that a shortest path degree of 10 may be considered relative to the rest of this dataset.

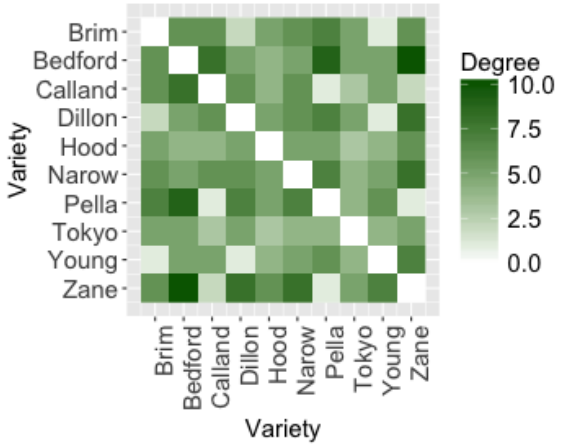

Figure 2.7: The shortest path degree matrix between ten varieties of interest.

2.11 Academic Genealogy of Statisticians

The `ggenealogy` package comes with two example datasets, and while we have introduced the plant breeding genealogy, we have yet to introduce the academic genealogy. As was demonstrated in Section 2.4, every parent in the academic genealogy is also a child, and some children in the academic genealogy have more than two parents. Neither of these features was the case in the plant breeding genealogy. Additionally, the academic genealogy is much larger than the plant breeding genealogy. Some of these differences may affect how one would approach `ggenealogy` plotting tools. For this reason, we will now demonstrate some of the `ggenealogy` plotting tools we already introduced, only now applied to the academic genealogy.

The ability to plot ancestors and descendants by generation was demonstrated using the plant breeding genealogy in Figure 2.6. As we believe this is the most novel plotting tool in the `ggenealogy` package, we will test it again here using the academic genealogy.

We need to choose a central individual of interest in order to create this plot. Perhaps we can use the academic statistician in the dataset that has the largest number of "descendants". To determine the name of this individual, below we use the `ggenealogy` function `getNode()` to create a vector `indVec` that contains the names of all individuals in the dataset. We then use the `dplyr` package to apply the `ggenealogy` function `getDescendants()` on each individual in the `indVec` vector (Wickham and Francois 2015). We set the parameter `gen` to a conservatively large value of 100 as this dataset is unlikely to have any individuals with more than 100 generations of "descendants".

After that, we can generate a table to examine all values of "descendant" counts in the dataset, along with the number of individuals who have each of those values of "descendant" counts. Of the 8165 individuals in this dataset, 6252 of them have zero "descendants", 322

of them have one "descendant", and 145 of them have two "descendants". There are only 17 individuals who have more than 30 "descendants", and there is one individual who has the largest value of 159 "descendants". We determine that this individual is the prominent British statistician Sir David Cox, who is known for the Box-Cox transformation and Cox processes, as well as for mentoring many younger researchers who later became notable statisticians themselves.

```
>R library(dplyr)
>R indVec <- getNodes(statGeneal)
>R indVec <- indVec[which(indVec != "", )]
>R dFunc <- function(var) nrow(getDescendants(var, statGeneal,
gen = 100))
>R numDesc <- sapply(indVec, dFunc)
>R table(numDesc)
```

```
numDesc
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
6252 322 145 88 58 36 31 22 23 14 17 13 14 9 9
15 16 17 18 19 20 21 22 23 24 25 26 27 29 30
6 4 3 2 5 7 5 3 3 2 2 6 1 1 3
34 37 38 40 41 44 45 48 49 61 62 75 77 84 159
2 1 1 1 1 1 1 1 2 1 1 1 1 1 1
```

```
R> which(numDesc == 159)
```

```
David Cox
1980
```

We can now visualize how these 159 "descendants" are related to Sir David Cox by calling the `plotAncDes()` function of `ggenealogy`, similar to what we did to generate Figure 2.6. As such, we create Figure 2.8 using the code below.

```
R> plotAncDes("David Cox", statGeneal, mAnc = 6, mDes = 6, vCol
= "blue")
```

We see from Figure 2.8 that Sir David Cox had 42 "children", many of them becoming notable statisticians themselves, such as Basilio Pereira, Valerie Isham, Gauss Cordeiro, Peter McCullagh, and Henry Wynn. Of his "children", the one who produced the most "children" of their own was Peter Bloomfield, who has 26 "children" and 49 "descendants". In total, Sir David Cox had five generations of academic statistics mentees in this dataset.

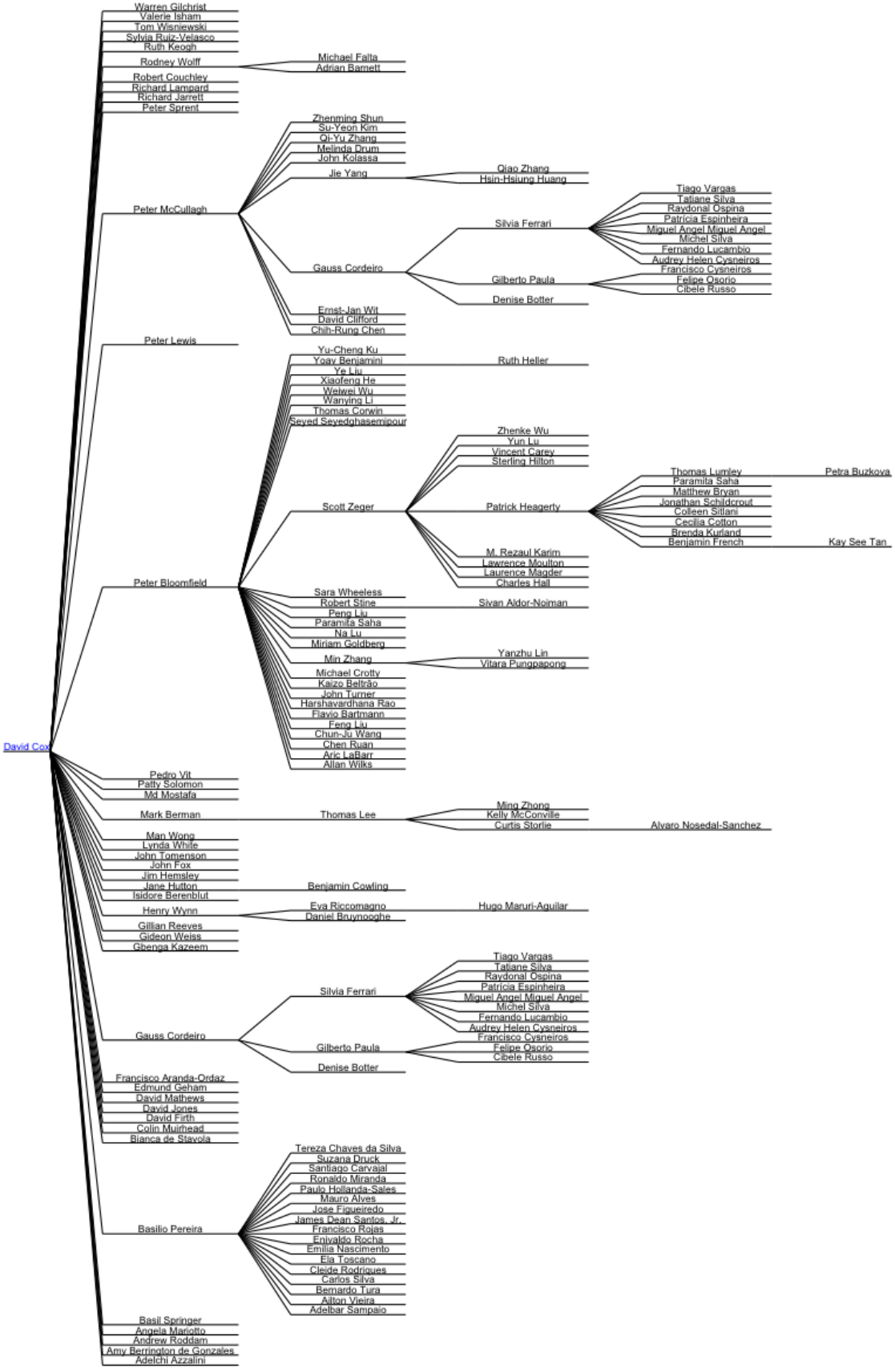

Figure 2.8: The 159 academic statistician "descendants" of Sir David Cox.

```
R> length(getChild("Peter Bloomfield", statGeneal))
```

```
[1] 26
```

```
R> nrow(getDescendants("Peter Bloomfield", statGeneal, gen =
100))
```

```
[1] 49
```

At this point, it would be insightful to examine a more detailed view of one of the longest strings of "parent-child" relationships between Sir David Cox and one of the two individuals who are his fifth generation "descendants". We do so with the code below, choosing his fifth generation "descendant" to be Petra Buzkova. We set the `fontFace` variable of the `plotPath()` to a value of 4, indicating we wish to boldface and italicize the two varieties of interest.

```
R> statIG <- dfToIG(statGeneal)
R> pathCB <- getPath("David Cox", "Petra Buzkova", statIG,
statGeneal, isDirected = FALSE)
R> plotPath(pathCB, fontFace = 4) + ggplot2::theme(axis.text
= ggplot2::element_text(size = 10), axis.title =
ggplot2::element_text(size = 10))
```

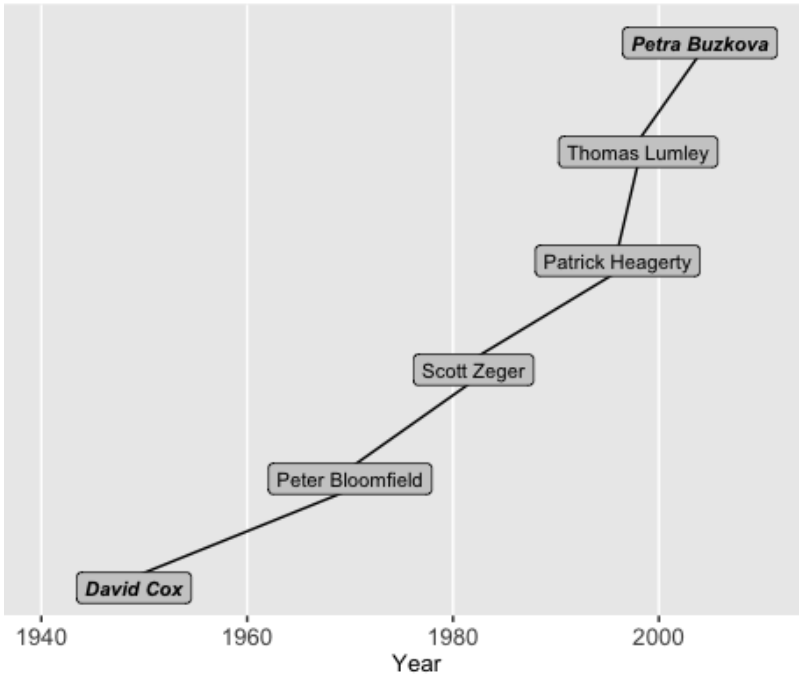

Figure 2.9: The shortest path between Sir David Cox and one of his fifth generation "descendants", Petra Buzkova.

This code results in Figure 2.9. We see that the shortest path between Sir David Cox and Petra Buzkova is strictly composed of five unidirectional "parent-child" relationships that span about 55 years. We see that the time difference between when an advisor and student earned their degrees is not consistent across this path: The three statisticians who earned their degrees earliest in this path span more than 30 years in degree acquisition, whereas the three statisticians who earned their degrees later in this path only span less than ten years in degree acquisition.

We also notice in Figure 2.9 that Sir David Cox received his statistics degree in about 1950, and Petra Buzkova received her statistics degree in about 2005. This genealogy only contains historical information about obtained degrees, and does not project into the future. Hence, we can be assured that Petra Buzkova is one of the younger individuals in the dataset, at least in the sense that the youngest individual could only have received his or her degree ten years after Petra Buzkova. However, we cannot be assured that Sir David Cox is one of the oldest individuals in the dataset. As such, it would be informative to superimpose this path of interest onto the entire dataset, using the `plotPathOnAll()` function of the `ggenealogy` package, as we did for the soybean genealogy in Figures 2.4 and 2.5.

We can achieve this using the below code. After trial and error, we use a `binVector` of size 200, and append `ggplot2` syntax to define suitable x-axis limits. The output of this

process is illustrated in Figure 2.10.

```
R> plotPathOnAll(pathCB, statGeneal, statIG, binVector =
1:200) + ggplot2::theme(axis.text = ggplot2::element_text(size
= 8), axis.title = ggplot2::element_text(size = 8)) +
ggplot2::scale_x_continuous(expand = c(.1, .2))
```

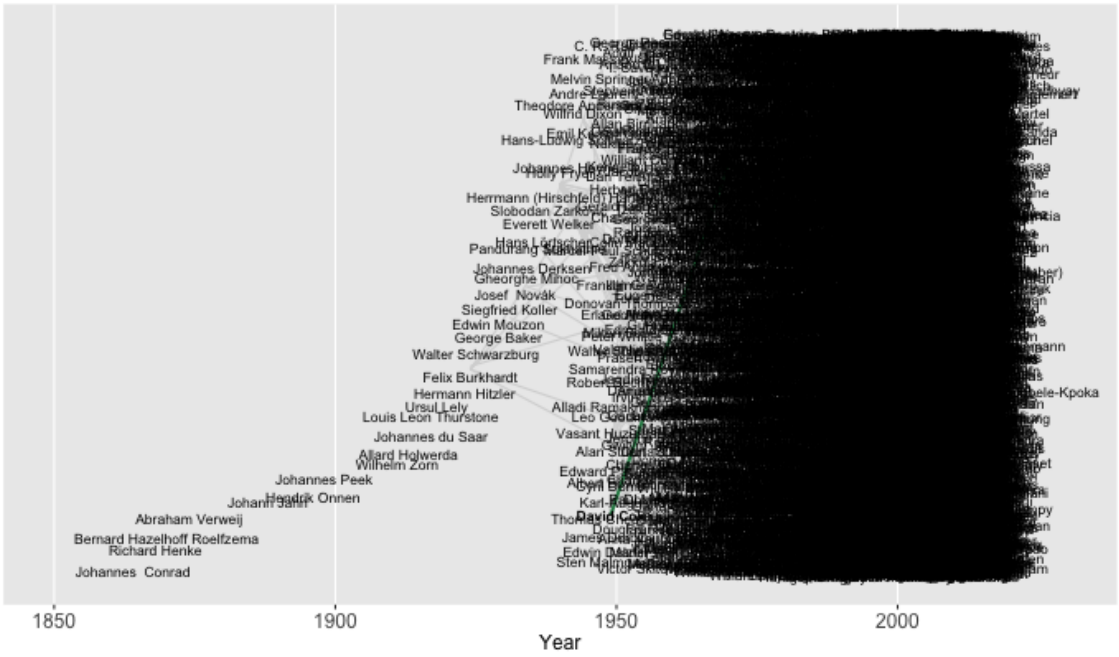

Figure 2.10: The shortest path between Sir David Cox and Petra Buzkova, superimposed over the data structure, using a bin size of 200.

We see from the resulting Figure 2.10 that almost all text labels for individuals who received their graduate-level statistics degrees between 1950 and 2015 are undecipherable. We also see that the year Sir David Cox acquired his statistics degree is somewhere in the later half of the variable year for this dataset, as the oldest dates for acquisition of statistics degrees in this dataset occur around 1860. However, the number of individuals who are documented as receiving their statistics degrees between 1860 and 1950 are few enough so that their text labels are somewhat readable.

The text labels are so numerous in Figure 2.10 that simply trying different values for the input parameter `binVector` will not solve the text overlapping problem. Instead, one approach we can try is to reconstruct the plot using the same `ggenealogy` function `plotPathOnAll()`, only now specifying variables to render the size (2.5) and color (default of black) of the text for nodes that are on the path of interest to be more noticeable than the size (0.5) and color (dark gray) of the text for nodes thare not on the path of interest. Moreover, we can make the edges that are not on the path of interest to be represented in

a less noticeable color (light gray) than the edges that are on the path of interest (default of dark green). The variable names and options for these aesthetics is further detailed in the help manual of the function. We provide one example code that alters the defaults of the text color and sizes of nodes and edges below, which results in Figure 2.11.

```
R> plotPathOnAll(pathCB, statGeneal, statIG,
binVector = 1:200, nodeSize = .5, pathNodeSize =
2.5, nodeCol = "darkgray", edgeCol = "lightgray") +
ggplot2::theme(axis.text = ggplot2::element_text(size
= 8), axis.title = ggplot2::element_text(size = 8)) +
ggplot2::scale_x_continuous(expand = c(.1, .2))
```

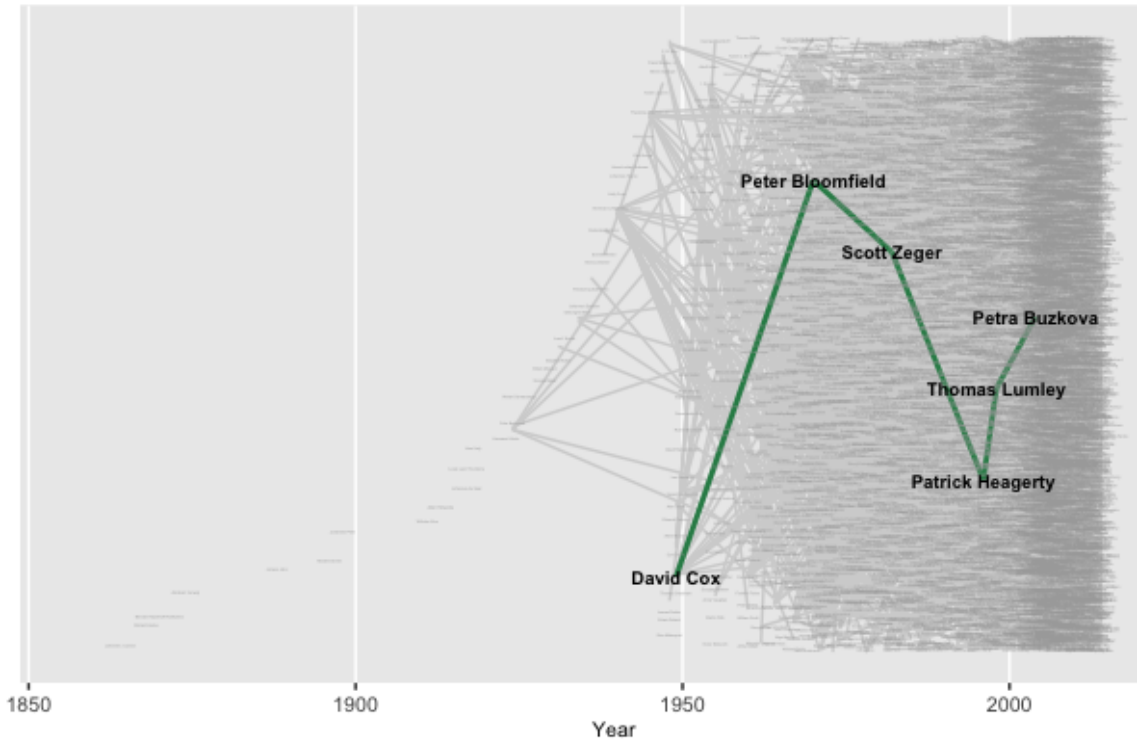

Figure 2.11: The shortest path between Sir David Cox and Petra Buzkova, superimposed over the data structure, using a bin size of 200. Individuals on the shortest path are labeled in large and black text and connected by dark green edges; all other individuals are labeled in small and gray text and connected by light gray edges.

In Figure 2.11, we can now see each individual on the path of interest, and how their values for the variable year are overlaid on the entire genealogy structure. We can also more clearly see that, even though only ten years span between the youngest individual in the genealogy and Petra Buzkova, there are many individuals in that last decade. Indeed, the decade from 2005 to 2015 appears to be the densest in this dataset in terms of acquisition

of statistics degrees.

2.11.1 Interactive visualization of genealogical structure

We could still improve upon Figure 2.11. Even though we may be primarily interested in understanding how the path of interest is overlaid across the entire genealogical structure, we could, upon viewing the entire structure, also develop an interest in nodes that are not on the path of interest but are revealed to stand out among the rest of the genealogical structure. For instance, in Figure 2.11, it may be of interest for us to determine the names of the few individuals who obtained their statistics degrees before 1900. Fortunately, within the `plotPathOnAll()` function, there is a variable `animate` that we can set to a value of TRUE to create an interactive version of the figure that allows us to hover over individual illegible labels and immediately receive their labels in a readable format. A short video demonstration of these interactive features can be viewed upon clicking on Figure **??**.

```
R> plotPathOnAll(pathCB, statGeneal, statIG, binVector =
1:200, nodeSize = .5, pathNodeSize = 2.5, nodeCol = "darkgray",
edgeCol = "lightgray", animate = TRUE)
```

2.12 Future Avenues

Incorporation of the Shiny application allows users to examine `ggenealogy` tools in a more interactive way (RStudio, Inc 2014). The reactive programming saves them the time of using command line for each change of input as well as the inefficiency of rerunning code. A Shiny application that uses certain `ggenealogy` functionality is available for users who wish to explore the soybean genealogy; the data can be viewed at `http://shiny.soybase.org/CNV/`.

We also aim to incorporate plotting tools that can examine not only quantitative variables (such as our example variable of "year"), but also categorical variables associated with individuals in datasets. Moreover, we look forward to testing the `ggenealogy` package on additional genealogical data sets. Exploring several datasets with the software will allow us to fix remaining bugs, and provide us further insight into how to make our tools available for a wide range of data input formats.

The `ggenealogy` visualization tool `plotPathOnAll()` is suitable as a data exploration tool, but not always as a publication tool. This is because we still see textual overlap in small-enough datasets (see Figure 2.5). As such, we plan to add a feature to the package that allows users to manually fine-tune automated plots. For example, after comparing several bin sizes on the soybean genealogy, we determined that the bin size of 6 produced the minimal textual overlap, as is seen in Figure 2.5. If we could subsequently fine-tune

the vertical positions of the small fraction of text labels that remained overlapped after the automated `ggenealogy` function, then we could potentially remove all overlaps, and the plot could be used in presentations and publications. Of course, it is impossible to eliminate textual overlap in larger datasets (see Figure 2.10). In such cases, we can remedy this problem by representing individuals who are not on the path of interest with dots instead of text (see Figure 2.11).

2.13 Conclusions

The `ggenealogy` package offers various plotting tools that can assist those studying genealogical lineages in the data exploration phases, as well as in preparing publication-suitable images. As each plot comes with its pros and cons, we recommended for users to explore several visualization tools. If users are simultaneously using similar packages, we in particular recommend using the `plotAncDes()` function. This plot allows users to view generation counts of a variety of interest in a manner that is not as readily available in similar software packages.

2.14 Acknowledgments

This project was an effort between Susan VanderPlas, Dianne Cook, Michelle A. Graham, and myself. Together, we thank Drs. James E. Specht and Randy C. Shoemaker for helpful discussions of soybean genealogy. In addition, we are grateful for the financial support from the United Soybean Board (Project 1204), The North Central Soybean Research Program, the NSF Plant Genome Research Program (award number 0820642), and the USDA-ARS CRIS Project 3625-21220-005-00D. The USDA is an equal opportunity provider and employer. Mention of trade names or commercial products in this article is solely for the purpose of providing specific information and does not imply recommendation or endorsement by the U.S. Department of Agriculture.

Chapter 3

Visualization methods for gene expression analysis

- Gene expression analysis is a relatively new application for statisticians. As with any new field of research the dizzying pace of development has created some confusion, forward advances and backward disappointments, and new but often immaturely developed methods. There is now a growing maturity to the field and a growing understanding of the primary purposes of gene expression studies, partly due to the increased involvement of statisticians. But it is still far from a fully-fledged discipline, and new issues continue to arise with the technology, analytical methods, software, and related data and information. This paper is about the surprises that may arise by plotting gene expression data, and how plots can reveal inadequacies in models and suggest ways to improve the models. When analyzing locally collected data, using conventional ANOVA methods, we found that the results didn't match findings that were obvious from data plots. We describe why this happened, and why it happens generally for similar types of data analyses, and the implications for ANOVA, p-values, false discovery rates and filtering of gene expression data (Cook et al. 2007).

- There are some commonly used graphics for gene expression data which are fraught with problems, and there are some less commonly used methods that are especially helpful. This paper discusses why this is, and ways to improve them (Cook et al. 2007).

- It is important to emphasize the difference between the analysis of gene expression data from many other multivariate data analysis tasks. In gene expression data it is important to find a small number of genes that are behaving differently to others in an understandable way. It is expected that most genes are expressing as normal, and only a few are expressing differently in response to a treatment. The expression values for each treatment are collected using a single microarray chip, which is read by an image reader to give numerical values for expression. These image values are calibrated across the chips, based on the assumption that the expression of most genes on all the chips is similar. The values on each chip are

logged because the distributions are skewed, and there is some evidence to suggest that intensity is exponentially distributed. Software such as Bioconductor (Gentleman et al. 2004) provides tools for normalizing microarray data and making plots to check the results of the normalization. When the calibration and normalization is complete, the task is to find the outliers, the most differently expressed genes, relative to the expression of most genes. Thus the task involves outlier detection

- It is also a multiple comparisons problem. From the perspective of a traditional statistical analysis we are merely dealing with a situation which could be solved by a t-test for comparing means. The drawback is that we have to do a test for every single gene! For example, the newer Affymetrix Arabidopsis ATH1 Genome Array chips have about 22000 genes. In calculating 22000 test statistics, according to a normal distribution we would expect to see at least 1100 genes with a significant p-value on a conventional level of 5%. That is simply by chance 1100 genes will emerge as interesting. This is not a practical approach to finding the few truly interesting genes to study in greater depth. These issues will be discussed later in the paper. This is the question of interest. Which genes are expressing differently in response to the experimental treatments? Difference is measured relative to the distribution of all the genes, and relative to consistent values in the replications. We need plots of the multivariate distributions to assess this. (Cook et al. 2007).

- Heatmaps are commonly-used but not helpful. Paper shows 2 clusters in heatmap, but it is really just one huge cluster in the scatterplot matrix (Cook et al. 2007). Scatterplot allows us to detect the outlying genes that could not be detected in the heatmap.

- The parallel coordinate plot (Inselberg 1985, Wegman 1990) is useful for examining relative patterns, such as those sought in gene expression data analysis. The axes are laid out in parallel rather than the orthogonal axes of scatterplots. These types of plots are commonly used to display gene expression data. Figure 4 displays the simulated data in parallel coordinates. What can we see from this plot? The pattern is a bow tie. The twist, between chips 2 and 3, means that the values of these chips are negatively correlated. The flat profiles between chips 1 and 2, and again between chips 3 and 4, mean that the values are positively correlated. Thus we get an overall picture that there is no strong clustering in this data. Yet, it is not a very clear picture because there are a lot of lines drawn here. Even though there are only 77 genes, which is tiny by microarray standards, the overplotting obscures important profiles such as the outlier. Commonly alpha-blending is used to alleviate the problem (Cook et al. 2007).

- Interaction on plots is critical! Especially important for gene expression analysis is the ability to probe and link plots. Exploring data depends on the ability to interact with a plot and link the changes to elements of other simultaneously visible displays (Becker and

Cleveland 1987, Newton 1978, McDonald 1982).

- With direct manipulation and linking the multivariate plots become very useful. Figure 5 shows the scatterplot matrix of the data from Figure 1 linked to parallel coordinates, generated with GGobi (Swayne et al. 2003, Lang et al. 2014, Cook and Swayne 2007). The outlier that is clearly visible in the scatterplot of chip 1 vs chip 2 is highlighted (orange or medium grey). This profile can now be seen in the profile plot, and it is obviously an outlier, on chip 1 relatively to it is expression on the other three chips. This is the simplest type of linking: the elements of all plots corresponding to the same row of the data matrix are all highlighted. This type of linking is commonly available in graphics software.

Summary overview of plot types

- Heatmaps provide an overview of the data. It is diffcult to grasp the distribution of values, because of a lack of geometric interpretability, diffculty in laying it out, and the mapping of numerical value to color. However, there is an appeal to heatmaps, and their use is very wide-spread. These plots can be enhanced to make them more useful. This is our wish list: * Different color mappings change the perception of patterns, and in exploratory analysis we want to see many different patterns to learn about different aspects of the data. We would like to add controls to allow the analyst to rapidly change the mapping of numerical value to color, and also the color scale. It would also be useful to switch from using color to glyphs (symbols) with size of glyph representing the the numerical value. * Methods to re-organize the rows and columns. Cluster analysis is one approach. For small subsets of data, manual controls like a large Rubik's cube, would be helpful. There is a package for R (Ihaka and Gentleman 1996) available on gclus, written by Catherine Hurley, which has some alternative approaches for rearranging the results of a dendrogram. Orca (Sutherland, Rossini, Lumley, Lewin-Koh, Dickerson, Cox and Cook 2000) has a few manual controls for rearranging rows and columns in a colored matrix. * The plot is not scalable, because at some point one reaches the maximum available plotting space. There need to be ways to reduce the size of a plot, such as spatial smoothing, and pan and zoom and slice controls for focusing on smaller subsets. * The heatmap needs to be probeable, to allow queries such as, "Which gene and which chip is it that has such a high expression value?" - Scatterplots are useful for pairwise comparisons, finding which genes are disproportionately expressed, and in tours they provide an overview of the multivariate distribution of expression values. Overplotting can be a problem for a large number of genes, but this can be alleviated using a density representation of the overplotted points. When there are many chips it is not possible to organize all pairs of plots into a scatterplot matrix. Linking between scatterplots is common and simple to implement. - Parallel coordinates can be used to overview data, but only when there are very few cases. Generally the large number of genes means that it is not possible to see the distribution of values using a parallel coordinate plot, although density plots based on alpha blending, such as those available in ExplorN 1, Mondrian and

Cassatt 2, can help. In the literature, and gene expression software packages, we usually see only subsets of the data plotted using parallel coordinates.

- These are a new type of plot, that we call a replicate line (RL) plot. We are now looking for genes that have a big difference between treatments (far from x = y), and that have little difference in replicates, small line length.

Chapter 4

Visualization methods for RNA-sequencing

The problem with only applying traditional modeling approaches to gene expression data is that these models are replete with assumptions that they alone cannot call into question. Fortunately, visualization enables analysts to see things they may not have expected with traditional modeling. By plotting the data, analysts can determine whether or not the applied models are sensible; they can work between the visualizations and the modeling, enhancing the models based on feedback from the plots. In sum, graphics are essential for analysts to check the quality of the data, assess the model diagnostics, and compare results from different methods.

Previous studies have provided sound evidence that gene expression data is most effectively explored by using graphical and numerical approaches in a complementary fashion (Cook et al. 2007). The authors of this study demonstrated this finding by applying modeling and visualization tools to simulated and real microarray data. Through the use of plots, they determined that their initial model assumptions were inappropriate and needed revision, and that the data quality was questionable and needed relabeling.

We would like to expand upon this knowledge that was originally applied to microarray data. In particular, we aim to develop, test, and publish visualization tools that are imperative for the exploration of the increasingly more common form of gene expression data - RNA-sequencing data. We tentatively plan to work on five main visualization techniques that can be applied to RNA-sequencing data.

- list 5 plot types here

4.1 Parallel coordinate plots

Side-by-side boxplots are often used to check the initial normalization process in RNA-sequencing pipelines (see Figure 4.1). When analyzing RNA-sequencing data, we only expect a handful of genes to show differential expression. As a result, when representing the read counts for each sample with a different boxplot, we expect the boxplots across the samples to look similar; that is, to have similar five number summaries and outlier distributions.

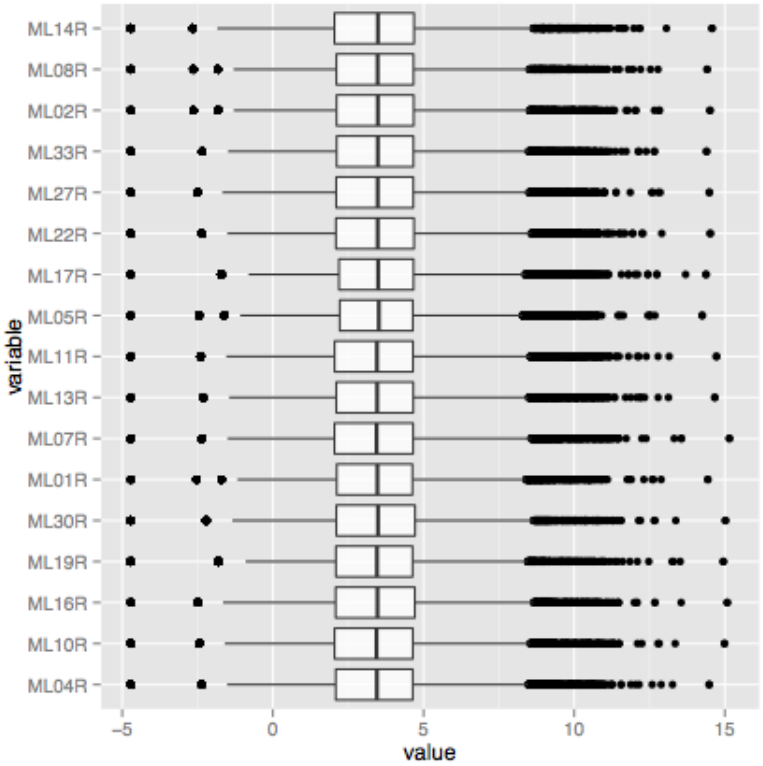

Figure 4.1: Boxplot of the 17 samples, after using RLE scaling factor method for normalization. Since we only expect a very small fraction of genes to show differential expression, the samples should appear similar after normalization. We can confirm that here, as the median values and IQR ranges are comparable across the samples.

Unfortunately, side-by-side boxplots can hide potential problems that could still be lurking in the data after normalization. This is because most RNA-sequencing normalization methods are conducted at the sample-level, and side-by-side boxplots do not show connections between the samples. Consequently, boxplots may deceive analysts to believe that the normalization succeeded in cases where alternative plots (such as parallel coordinate plots) would have otherwise indicated that the normalization was in fact unsuccessful due to the presence of inconsistencies (crossings) between replicates.

For this reason, parallel coordinate plots are an essential visual tool when checking the

initial normalization process. If the normalization is adequate, then the connections between replications should be flat, and we should only see crossings between treatement groups. An example of a parallel coordinate plot showing successful normalization is shown in Figure 4.2.

"The pattern is a bow tie. The twist, between chips 2 and 3, means that the values of these chips are negatively correlated. The flat profiles between chips 1 and 2, and again between chips 3 and 4, mean that the values are positively correlated. Thus we get an overall picture that there is no strong clustering in this data. Yet, it is not a very clear picture because there are a lot of lines drawn here."

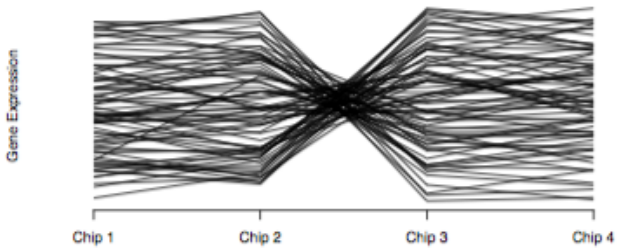

Figure 4.2: Simulated microarray data showing two treatment groups

The tools currently available for constructing parallel coordinate plots to assess RNA-sequencing data are severely limited: Namely, the large number of genes present can cause constraints in space and time. With one line being drawn for each gene, our resulting plots will have too many lines being drawn to even view patterns of interest in the first place. Moreover, the construction of these plots can be consuming in computation and time, often requiring analysts to wait for minutes before they can view the output.

Even when popular tools to visualize parallel coordinate plots (such as the `ggparcoord` function in the `GGally` package) can be very useful for many datasets, they are not always helpful for RNA-sequencing datasets (Schloerke et al. 2016). As a result, our goal is to develop a new approach that both quickly and meaningfully displays the key pieces of information for RNA-sequencing data with parallel coordinate plots.

4.2 Pairwise scatterplots

We aim to develop plots that can effectively plot replicates against each other. We can do this by generating pairwise scatterplots for replicates, using the `scatmat()` function of the `GGally` package (Schloerke et al. 2016). If the data is high quality, then we would expect negligent differences between read counts across replicates. This would result in a

scatterplot where the read counts from the two replicates fall along the x=y relationship. Deviations from this expectation would indicate a quality problem with the data.

With this second goal we face the same challenges that we did with the first goal: We have a plotting algorithm that is not tailored to deal with the vastly sized datasets like RNA-sequencing data. As a result, we again face space (overplotting) and time (computational speed) constraints, and our goal would be to tailor the scatterplot matrix to adapt to large amounts of data.

In addition, we must now recognize that the few data points along the perimeter of the x=y relationship that we can see may be misleading. Hence, we would want to develop guidelines in terms of what denotes a high-quality dataset where read counts are similar between replicates and different between treatments.

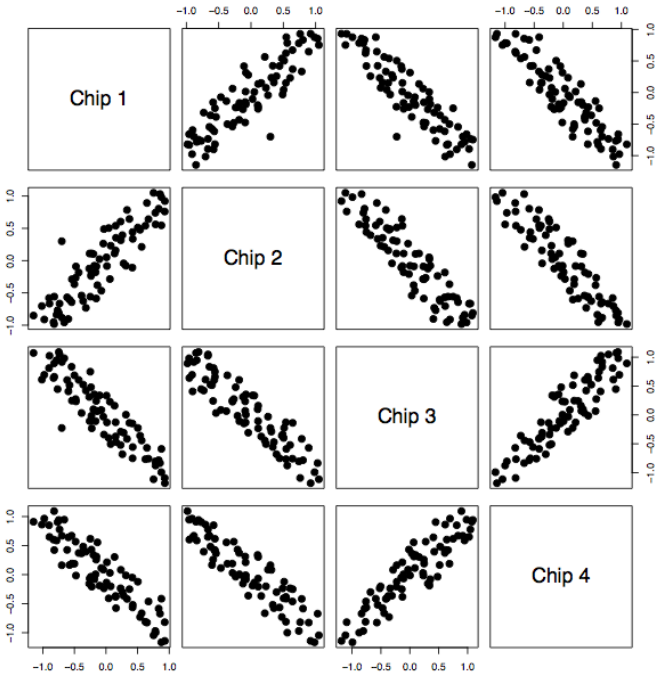

Figure 4.3: Simulated scatterplot matrix

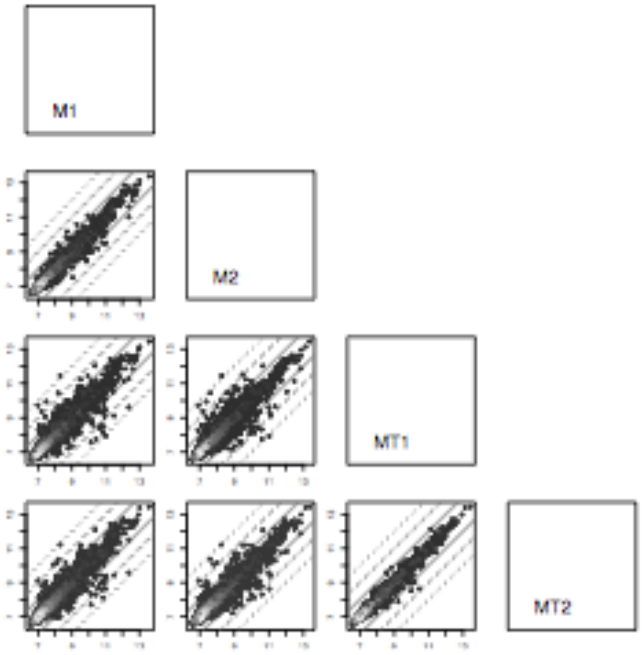

Figure 4.4: Simulated scatterplot matrix 2

4.3 Replicate line plots

third thing is want to be able to see replicates (scatterplot of treatment 1 versus treatment 2) and connect replicates (porcupine plots, but they only work for 2 reps) (but have that kind of display for >2 reps. what is variability in replicate versus variability across treatments)

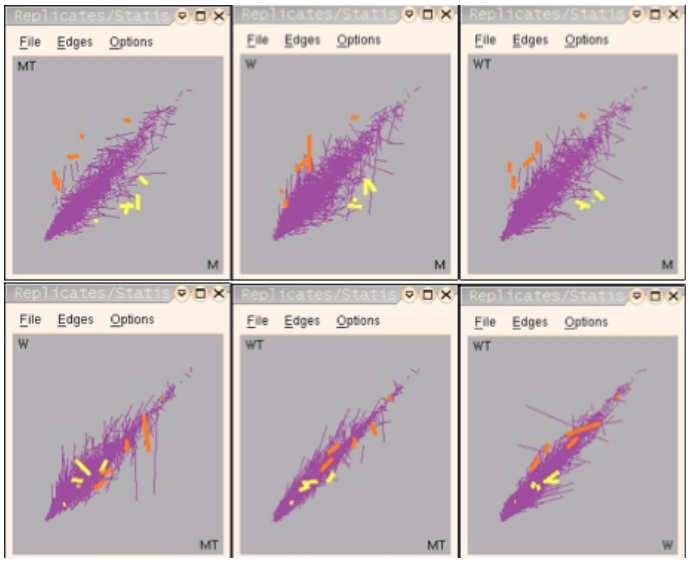

Figure 4.5: Porcupine plot

4.4 Single gene plots

fourth thing is want interaction plots that allow us to easily draw single gene plots of top candidates.

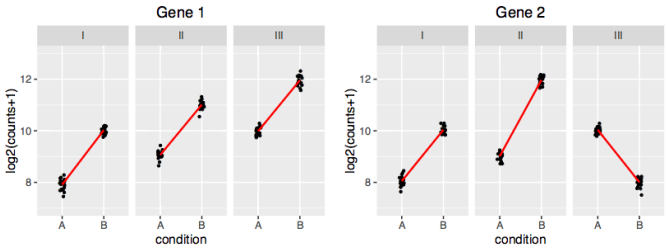

Figure 4.6: Single gene plot

4.5 Permutation testing plots

fifth thing is permutation testing. that should be a part of this package. we want to see what this might look like if we permuted (get confidence bands on what we might see if it is just random noise as the difference). want to build confidence bands on the visualization based on the permutation ideas.

- triplets - generate display with 3 plots (one is actual data, other two are perm1, perm2). for the most sig genes - do top 20 genes as triplets. randomize the location of the actual data amongst the other two. (so don't know which one is permuted and which one is not permuted) - this is line-ups (triangle-test, b/c only three), and so can use the package (nullabor). have the three along a row. top gene under permuted design versus real data.

Chapter 5

Stock syntax

5.1 Database structure

As an example, the learning outcomes for `Topic 03` are provided in List 1 below.

Now, we will run the REFER TO SECTION .

Table 5.1: Topic numbers and descriptions

Number	**Description**
01	Data
02	Descriptive Statistics for a Single Categorical Variable
03	Descriptive Statistics for a Single Quantitative Variable

List 1: Learning outcomes for Topic 03

A. Use standardizing to determine how many standard deviations an observation is away from the mean value.

B. Use z-scores to compare observations for different quantitative variables.

C. Explain how standardizing affects the shape, center, and variability of the distribution of a quantitative variable.

Example question title: **T16.A.A.04-1.1.MC.1**

The absolute pathway to the `extdata` directory on your local computer can be determined by typing the following command into the **R** console:

```
system.file("inst/extdata/", package = "ePort")
```

```
keyHTM = system.file("inst/extdata/KeyFiles/Topic06.Questions.htm", package =
  "ePort")

refineKey(keyHTM)

keyPath = gsub("htm$", "txt", keyHTM)

dataPath = system.file("inst/extdata/DataFiles/Topic06/Topic06.A.csv", package
  "ePort")

rewriteData(dataPath)

loPath = system.file("inst/extdata/LOFiles/Topic06.Outcomes.txt", package = "e

outPath = system.file("inst/extdata/OutputFiles", package = "ePort")

makeReport(keyFile = keyPath, dataFile = dataPath, loFile = loPath, outFile =
  outPath)
```

```
merged = subsetData(mergedData, dataTable)
makeReport(outFile = outPath, unit = 2, reportType = "crossSecUnit", className
  "Eng444", repeatLowScore = 70)
```

- One topic for one section - short version ("secTopicShort")
- One topic for one section - long version ("secTopicLong")
- One topic comparing multiple sections - short version ("crossSecTopicShort")
- One topic comparing multiple sections - long version ("crossSecTopicLong")
- One unit (group of topics) for one section ("secUnit")
- One unit (group of topics) comparing multiple sections ("crossSecUnit")

Bibliography

Richard A. Becker and William S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2): 127–142, 1987.

Dianne Cook and Deborah F. Swayne. *Interactive and Dynamic Graphics for Data Analysis.* Springer, 2007.

Dianne Cook, Heike Hofmann, Eun-Kyung Lee, Hao Yang, Basil Nikolau, and Eve Wurtele. Exploring gene expression data, using plots. *Journal of Data Science*, 5:151–182, 2007.

Albart Coster. *pedigree: Pedigree functions*, 2013. URL https://CRAN.R-project.org/package=pedigree. R package version 0.4.

Gabor Csardi and Tamas Nepusz. The `igraph` software package for complex network research. *InterJournal, Complex Systems*, 1695, 2006. URL http://igraph.sf.net.

Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*, 30(11): 1203–1233, 2000.

Robert C. Gentleman, Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean Y.H. Yang, and Jianhua Zhang. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5(10):R80, 2004.

Theodore Hymowitz, Christine A. Newell, and Samuel G. Carmer. *Pedigrees of Soybean Cultivars Released in the United States and Canada.* International Soybean Series, College of Agriculture, University of Illinois at Urbana-Champaign, Urbana, IL, 1977.

Alfred Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1:69–91, 1985.

Duncan T. Lang, Debby Swayne, Hadley Wickham, and Michael Lawrence. *rggobi: Interface between R and GGobi*, 2014. URL https://CRAN.R-project.org/package=rggobi. R package version 2.1.20.

John A. McDonald. *Interactive Graphics for Data Analysis*, 1982. Technical Report Orion II, Statistics Department, Stanford University.

C. Newton. Graphica: From alpha to omega in data analysis. In P.C.C. Wang, editor, *Graphical Representation of Multivariate Data*, pages 59–92. Academic Press, New York, NY, 1978.

North Dakota State University and American Mathematical Society. *The Mathematics Genealogy Project*, 2010. URL http://www.genealogy.math.ndsu.nodak.edu. Archived Web Site. Retrieved from the Library of Congress, Accessed on March 6, 2015.

PostgreSQL, 2016. URL http://www.postgresql.org/.

RStudio, Inc. *shiny: Web Application Framework for R*. 2014. URL http://CRAN.R-project.org/package=shiny. R package version 0.10.2.1.

Lindsay Rutter, Susan Vanderplas, and Dianne Cook. *ggenealogy: Visualization Tools for Genealogical Data*, 2015. URL https://CRAN.R-project.org/package=ggenealogy. R package version 0.1.0.

Barret Schloerke, Jason Crowley, Di Cook, Francois Briatte, Moritz Marbach, Edwin Thoen, and Amos Elberg. *GGally: Extension to ggplot2*, 2016. URL https://CRAN.R-project.org/package=GGally. R package version 1.0.1.

Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, Jonathan T. Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, 2003.

Deborah F. Swayne, Duncan T. Lang, Andreas Buja, and Dianne Cook. Ggobi: Evolving from xgobi into an extensible framework for interactive data visualization. *Journal of Computational Statistics and Data Analysis*, 43:423–444, 2003.

Terry Therneau, Schaid Daniel, Jason Sinnwell, and Elizabeth Atkinson. *kinship2: Pedigree Functions*, 2015. URL https://CRAN.R-project.org/package=kinship2. R package version 1.6.4.

Edward J. Wegman. Hyperdimensional data analysis using parallel coordinates. *Journal of American Statistics Association*, 85:664–675, 1990.

Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, New York, NY, 2009. URL http://had.co.nz/ggplot2/book.

Hadley Wickham and Romain Francois. *dplyr: A Grammar of Data Manipulation*, 2015. URL https://CRAN.R-project.org/package=dplyr. R package version 0.4.3.