Journal of Statistical Software

May 2019, Volume 89, Issue 13. doi: 10.18637/jss.v089.i13

ggenealogy: An R Package for Visualizing Genealogical Data

Lindsay Rutter
Iowa State University

Susan VanderPlas
Iowa State University

Dianne Cook
Monash University

Michelle A. Graham
USDA Agriculture Research Service

Abstract

This paper introduces **ggenealogy** (Rutter, Vanderplas, and Cook 2019), 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, parsing and performing calculations on branches of interest, and displaying the results. It is possible 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. The tools are being tested on a dataset with milestone cultivars of soybean varieties (Hymowitz, Newell, and Carmer 1977) as well as on a web-based database of the academic genealogy of mathematicians (North Dakota State University and American Mathematical Society 2010). The latest stable package version is available in source and binary form on the Comprehensive R Archive Network (CRAN).

Keywords: genealogy, data visualization, statistical graphics, exploratory data analysis, interactive, R.

1. 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** (Rutter *et al.* 2019) 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 2010).

2. Available software

Publishing in the open source R statistical programming language (R Core Team 2019) 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, Daniel, Sinnwell, and Atkinson 2015). In Figure 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 many 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

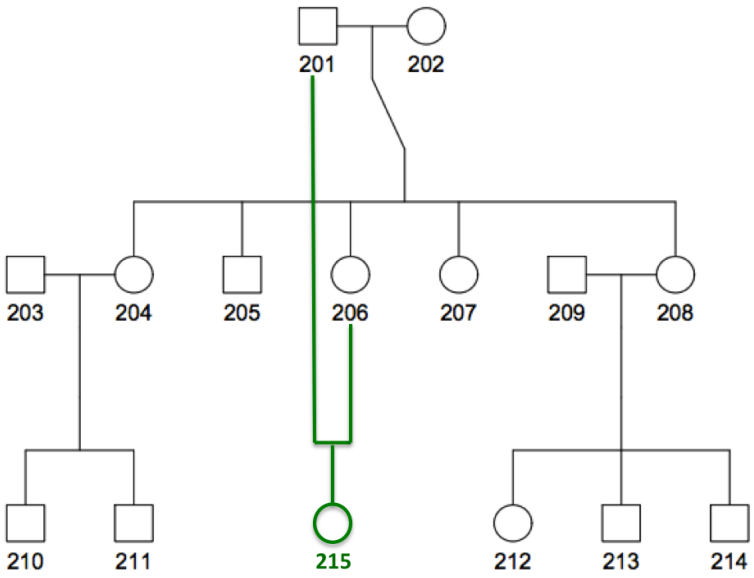

Figure 1: Example pedigree chart from the **kinship2** package, where the vertical axis denotes generation count. We superimposed green-highlighted individual 215 for explanatory purposes. As an offspring of a parent-child relationship, individual 215 is both a second and third generation individual. Hence, it should be displayed twice on the vertical axis, once for each of its generation counts. However, most standard pedigree tools only allow for an individual to be displayed once. In the **kinship2** package, individuals can indeed be displayed more than once. However, each child must have zero or two parents (one male and one female). These restrictions make it impossible to plot genealogical data by generation count in cases where there are many inter-generational breedings.

case in Figure 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, most 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, most standardized tools for visualizing genealogical information ambiguously portray the genealogical dataset by generation count.

In the **kinship2** package, if an individual cannot be represented with only one instance, then it will be completely copied and connected with dotted lines to the relevant individuals. However, the package requires that each child has exactly zero or two parents; if a child has two parents, then one must be female and one must be male. These requirements preclude certain genealogical datasets from being plotted by generation count, especially when their complexity increases with inter-generational breeding.

In addition, popular graph drawing software such as **GraphViz** and **Cytoscape** can be used

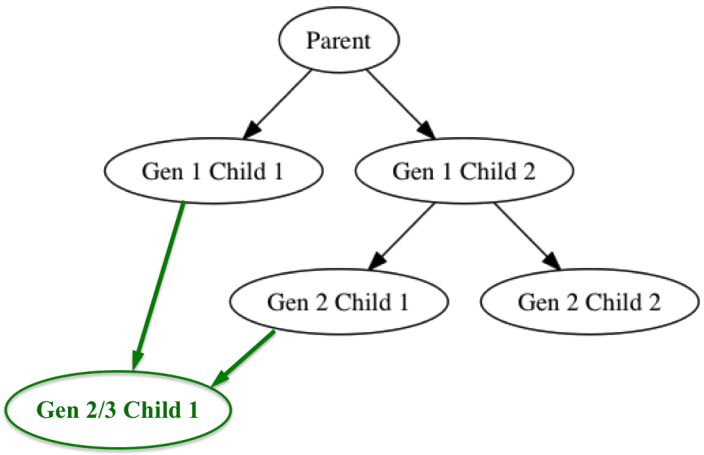

Figure 2: Example genealogical display using popular graph software like **GraphViz** and **Cytoscape**, with generation count denoted by the vertical axis. As was shown in Figure 1, the green node has parents from two different generations, and hence must be ambiguously positioned as one of two generation counts.

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 packages cannot precisely portray the genealogical dataset in cases of inter-generational breeding.

We again illustrate this problem in Figure 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 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 9, we will demonstrate **ggenealogy** plots that can remedy these problems.

3. Package overview

We will now provide a brief overview of the functionality of the **ggenealogy** package before going into more detail with examples later in this paper. With the **ggenealogy** package, users can convert genealogical data into graph structures. This allows for users to efficiently traverse, analyze, and elicit graph theoretical measurements on genealogical lineages (see Section 6). These capabilities were developed by building upon the **igraph** package (Csardi and Nepusz 2006).

Additionally, the **ggenealogy** package allows for users to plot genealogical data in multiple ways. Users can obtain and plot the shortest path between two nodes of interest constrained on a variable of interest (see Section 7), superimpose a shortest path between two nodes of interest

with the entire genealogical structure constrained on a variable of interest (see Section 8), plot the ancestors and descendants of a node of interest constrained on generation count (see Section 9), and plot distance matrices based on a variable of interest (see Section 10). Most of these plotting tools were developed by building upon the **ggplot2** package (Wickham 2009). As such, most of these plotting tools can be customized by appending syntax from the **ggplot2** package, as will be demonstrated in example code throughout this paper. Moreover, some of these plotting tools have interactive capabilities that were developed with the use of the **plotly** package (Sievert *et al.* 2019, see Section 12).

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 in this paper to demonstrate some of the tools available in the software.

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> library("ggenealogy")
R> library("dplyr")
```

```
R> data("sbGeneal", package = "ggenealogy")
R> str(sbGeneal)

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

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 (North Dakota State University and American Mathematical Society 2010), 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 2016).

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 post-baccalaureate 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", package = "ggenealogy")
R> dim(statGeneal)

[1] 8165    6

R> colnames(statGeneal)

[1] "child"    "parent"   "gradYear" "country"  "school"   "thesis"

R> statGenealEP <- statGeneal %>% filter(parent != "")
R> statIG <- dfToIG(statGenealEP)
R> uCP <- na.omit(c(statGeneal$child, statGeneal$parent))
R> length(unique(uCP[uCP != ""]))

[1] 7122

R> uChild <- unique(na.omit(statGeneal$child))
R> uParent <- unique(na.omit(statGeneal$parent))
R> nrow(na.omit(summarise(group_by(statGeneal,child))))

[1] 7122

R> nrow(na.omit(summarise(group_by(statGeneal,parent))))

[1] 872

R> table(summarise(group_by(statGenealEP,child),
+    cPC = sum(!is.na(parent)))$cPC)

   1    2    3    4
2801  226   10    2

R> getBasicStatistics(statIG)$numEdges

[1] 3291
```

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 quantitative variable column. Each case in this quantitative variable column should be of type numeric. In the first example dataset, columns that could be used for this purpose include "devYear", "yield", and "yearImputed"; in the second example dataset, the column that could be used for this purpose is "gradYear". However, for these quantitative variable columns to successfully be used in plots, we would first need to assure that each row within them contains a numeric value (not `NA`). We could achieve this by filtering out or imputing certain rows that contain `NA` values for this column of interest. We demonstrate a filtering process for this purpose at the end of Section 7. After that, any **ggenealogy** plot can be used.

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 dataset of soybean genealogy.

We now convert it into an **igraph** object `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
```

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 path, along with the variety name and quantitative variable value of interest for each variety in the path. For this example, we will use the development year (from the column "devYear") as our quantitative variable.

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

$pathVertices
[1] "Tokyo"    "Volstate" "Jackson"  "R66-873"  "Narow"
$variableVertices
[1] "1907"   "1942"   "1954.5" "1971.5" "1985"
```

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

```
R> plotPath(pathTN, sbGeneal, "devYear")
```

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 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

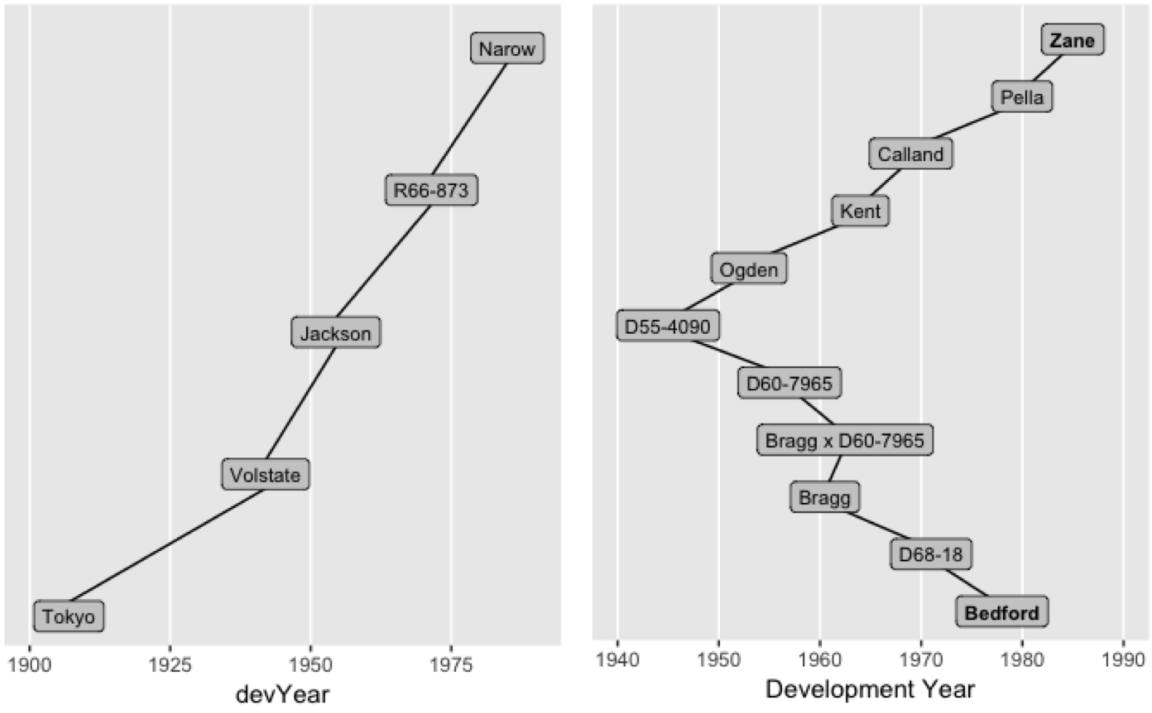

Figure 3: Left: The shortest path between varieties `Tokyo` and `Narow` is strictly composed of a unidirectional sequence of parent-child relationships. Right: The shortest path between varieties `Zane` and `Bedford` is not strictly composed of unidirectional parent-child relationships; they instead have a cousin-like relationship.

parent-child relationships, with `Tokyo` as the starting ancestor in the early 1900s, `Narow` as the most recent descendant 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, we can call the **ggenealogy** function `getVariable()` using the input quantitative variable of development year. This indicates that variety `Bedford` was released in 1978 and variety `Zane` in 1985.

```
R> getVariable("Bedford", sbGeneal, "devYear")

[1] 1978

R> getVariable("Zane", sbGeneal, "devYear")

[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()` function to a value of 2, indicating we wish to boldface the two varieties of interest. In addition, as is the case with plotting tools in **ggenealogy**, we can append **ggplot2** syntax. In this case, we may wish to hard code the x-axis label from its default value of "devYear" (the inputted quantitative variable column name) to the more readable "Development Year".

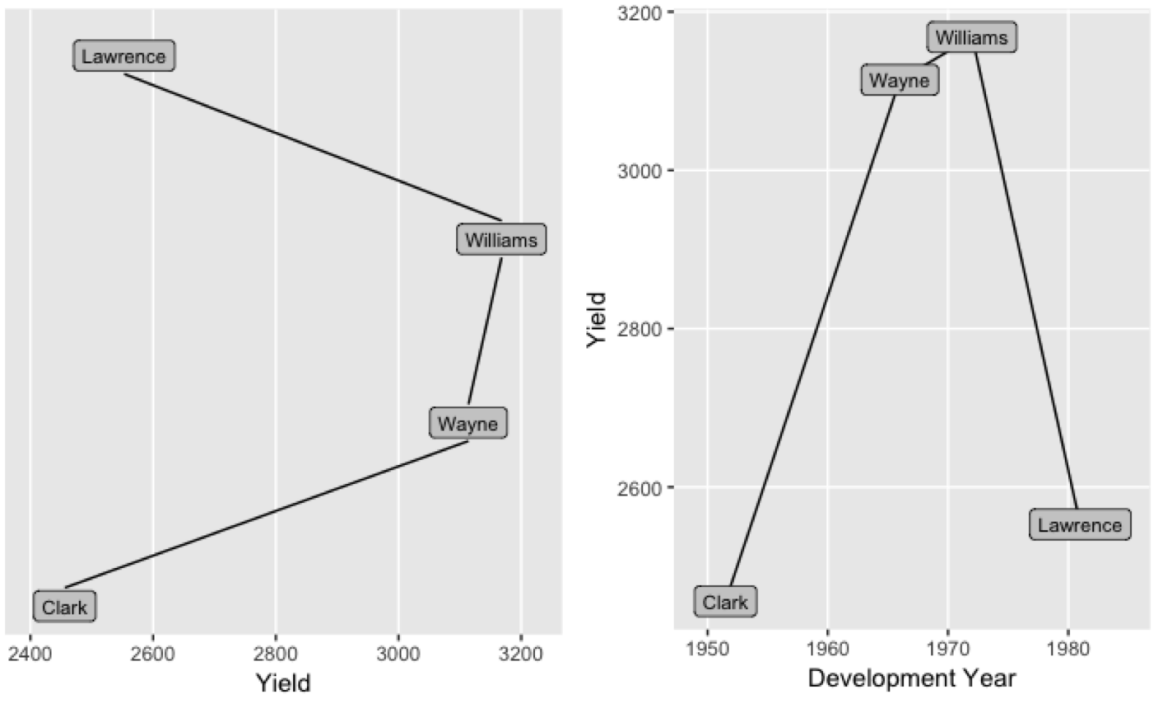

Figure 4: Left: Plotting a path of interest using a new quantitative variable of interest, "Yield". Right: Plotting a path of interest using two quantitative variables. We see that the varieties `Clark` and `Lawrence` have lower yields than the varieties in the middle of the path.

```
R> pathBZ <- getPath("Bedford", "Zane", sbIG, sbGeneal, "devYear")
R> plotPath(pathBZ, sbGeneal, "devYear", fontFace = 2) +
+    ggplot2::xlab("Development Year")
```

The resulting plot (right half of Figure 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 seen in 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 that differentiates `Zane` and `Bedford` (protein content, yield, disease resistance, etc.) can also be examined across these two separate lineage histories.

We would like to note that the `plotPath()` function can be used with one or two quantitative variables that the user hard-codes. We illustrate this with the `sbGeneal` data frame after we subset it to remove observations that do not have values for the quantitative variable "yield". Then, we can examine the path between the varieties `Clark` and `Lawrence` and plot how the quantitative variable "yield" changes along the parent-child relationships of that path. Likewise, we plot how the quantitative variables "devYear" and "yield" both change along the parent-child relationships of that path. The output of these two calls to the `plotPath()` function can be viewed in Figure 4.

```
R> sbFilt <- sbGeneal[complete.cases(sbGeneal[1:3]), ]
R> sbFiltIG <- dfToIG(sbFilt)
R> pathCL <- getPath("Clark", "Lawrence", sbFiltIG, sbFilt, "yield")
R> plotPath(pathCL, sbFilt, "yield") + ggplot2::xlab("Yield")
R> pathCL2 <- getPath("Clark", "Lawrence", sbFiltIG, sbFilt, "devYear")
R> plotPath(pathCL2, sbFilt, "devYear", "yield") +
+    ggplot2::xlab("Development Year") + ggplot2::ylab("Yield")
```

8. Superimposing a shortest path on a 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? 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 5.

```
R> plotPathOnAll(pathTN, sbGeneal, sbIG, "devYear", bin = 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. While the first four explicit parameters have been introduced earlier in this paper, the fifth parameter (`bin`) 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, development year). 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 (unless a user specifies a second quantitative variable of interest, as we will demonstrate later). 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 (`bin`).

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

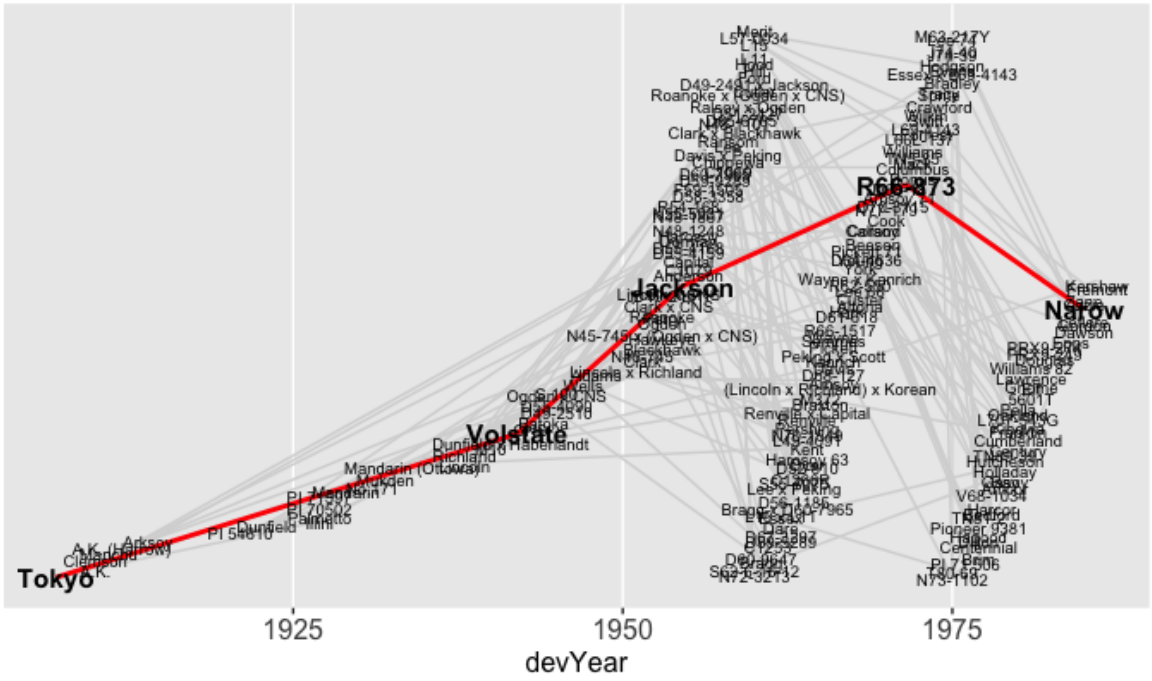

Figure 5: The shortest path between `Tokyo` and `Narow`, superimposed over the data structure, using a bin size of 3.

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, `bin` 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 5, 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 5 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 5 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 6.

```
R> plotPathOnAll(pathTN, sbGeneal, sbIG, "devYear", bin = 6, pathEdgeCol =
+    "seagreen2", nodeSize = 1, pathNodeSize = 3) +
+    ggplot2::xlab("Development Year")
```

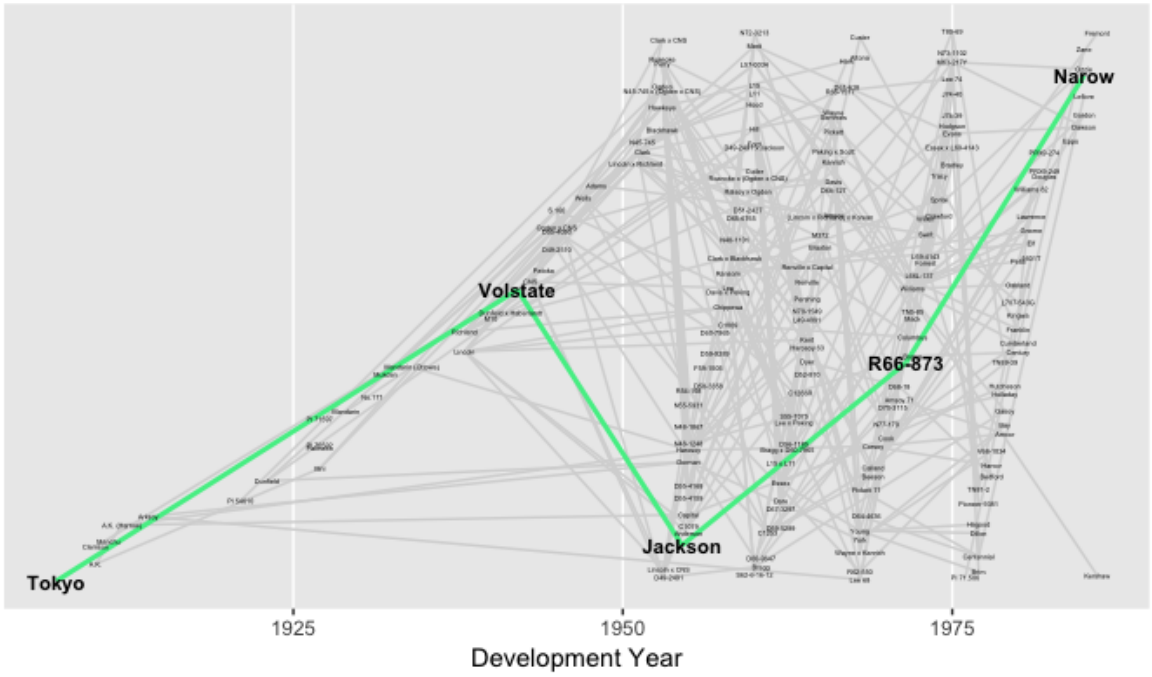

Figure 6: The shortest path between `Tokyo` and `Narow`, superimposed over the data structure, using a bin size of 6.

We can immediately see that Figure 6 more successfully mitigates text overlap compared to Figure 5. We also confirm what we saw in the previous plot that most varieties were released between 1950 and 1970, and any textual overlap is confined to this range of years.

Notice again from Figure 5 that the default horizontal axis label for the `plotPath()` method has a value of "devYear". We wanted to change the default value of the horizontal axis label to "Development Year". We did this in the code above for Figure 6 by appending the `ggplot2::xlab()` function.

We would like to emphasize that the `plotPathOnAll()` function can be used with one or two quantitative variables that the user hard-codes. We demonstrate this using the filtered data frame and **igraph** objects (`sbFilt` and `sbFiltIG`) to assure that there were no `NA` values in the quantitative variable "yield". Then, we can examine the path across the remaining genealogical structure and how the quantitative variable "yield" changes along the parent-child relationships of that path. Likewise, we can plot how the quantitative variables "devYear" and "yield" both change along the parent-child relationships of that path. The output of these two calls to the `plotPathOnAll()` function can be viewed in Figure 7.

```
R> plotPathOnAll(pathCL, sbFilt, sbFiltIG, "yield", bin = 3, pathEdgeCol =
+    "purple") + ggplot2::xlab("Yield")
R> plotPathOnAll(pathCL, sbFilt, sbFiltIG, "yield", "devYear",
+    pathEdgeCol = "orange") + ggplot2::xlab("Yield") +
+    ggplot2::ylab("Development Year")
```

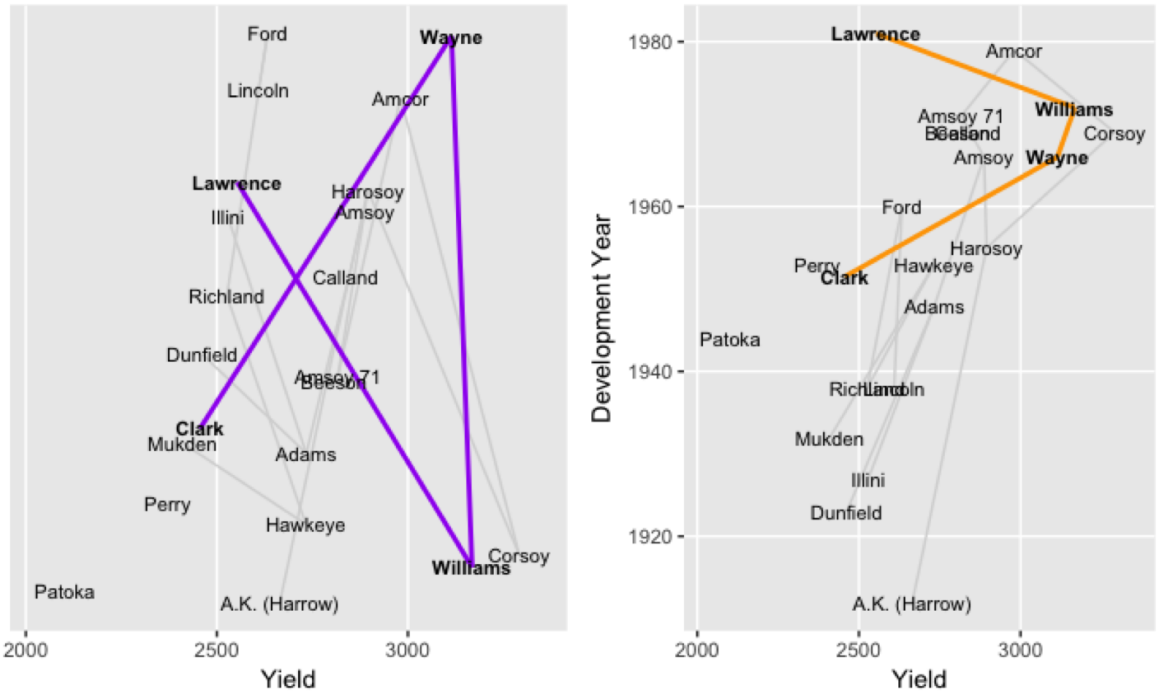

Figure 7: Left: Plotting a path of interest across the genealogical structure using a new quantitative variable of interest, "Yield". Right: Plotting a path of interest across the genealogical structure using two quantitative variables. We see that an increase in development year might correlate with an increase in yield for the varieties in this dataset, and that our path of interest is composed of generally younger varieties compared to the dataset as a whole.

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 uniquely beneficial because most genealogy and graph visualization software packages do not allow for repeated node labels even though some genealogical datasets require repeated node labels in order to be visualized by generation counts (as was shown in Figures 1 and 2).

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 8. We notice that `Lee` has 3 generations of ancestors and 5 generations of descendants. We also notice that some varieties are repeated, which is a unique feature provided by **ggenealogy**. For example, the variety `5601T` is represented four

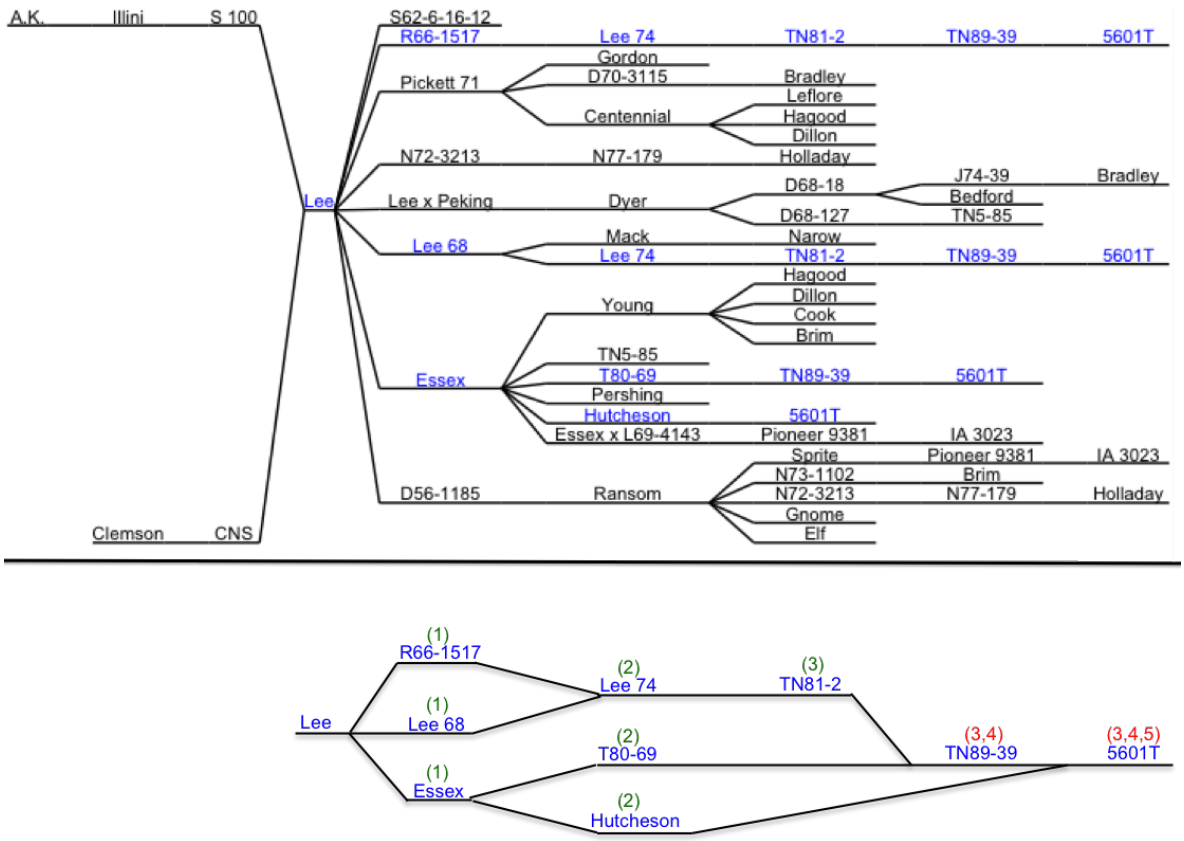

Figure 8: Top: All ancestors and descendants of the variety `Lee` are shown in this **ggenealogy** plot. Bottom: For didactic purposes, this plot was constructed manually outside of the **ggenealogy** package. It mimics the blue paths in the **ggenealogy** plot on the top, only now nodes cannot be repeated. The parenthetical numbers above each node represent the set of generation counts that node is away from the center node `Lee`. The presence of red parentheses indicates that the plot on the bottom ambiguously displays the example soybean genealogy in contrast to the way that the **ggenealogy** plot on the top can accomplish.

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.

The bottom plot of Figure 8 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 descendant of node `Lee`, and node `5601T` cannot simultaneously be represented as a third, fourth, and fifth descendant 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.

10. Plotting a distance matrix

It may also be of interest to generate matrices where the colors indicate 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 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) +
+    ggplot2::scale_fill_continuous(low = "white", high = "darkgreen") +
+    ggplot2::theme(legend.title = ggplot2::element_text(size = 15),
+    legend.text = ggplot2::element_text(size = 15)) +
+    ggplot2::labs(x = "Variety", y = "Variety")
```

This creates the plot in Figure 9. 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 3. However, we now also see that a shortest path degree of 10 may be considered relative to the rest of this dataset.

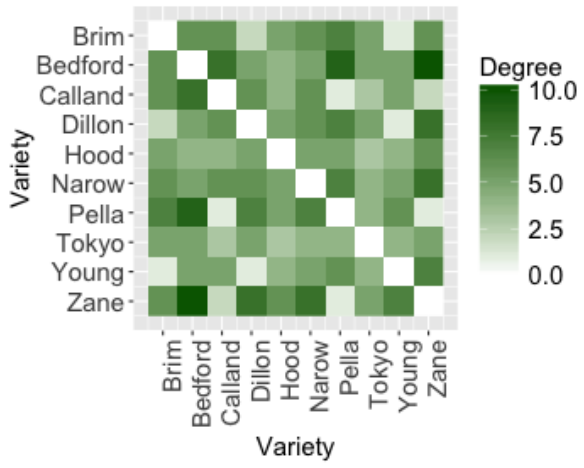

Figure 9: The shortest path degree matrix between ten varieties of interest.

11. Academic genealogy of statisticians

An academic genealogy is the second dataset provided in the package, where every parent is also a child and some children have more than two parents, as was briefly described in Section 4. Neither of these features were present 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 8. 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, François, Henry, and Müller 2019). 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, 6251 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
6251  322  145   88   58   36   31   22   23   14   17   13   14   10    9
  15   16   17   18   19   20   21   22   23   24   25   26   27   29   30
   6    4    3    2    5    7    4    3    3    2    2    6    1    1    3
  34   37   38   40   41   44   45   48   49   60   61   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 8. As such, we create Figure 10 using the code below.

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

We see from Figure 10 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.

```
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 statisticians of interest.

```
R> statIG <- dfToIG(statGeneal)
R> pathCB <- getPath("David Cox", "Petra Buzkova", statIG, statGeneal,
+    "gradYear", isDirected = FALSE)
R> plotPath(pathCB, statGeneal, "gradYear", fontFace = 4) +
+    ggplot2::xlab("Graduation Year") + ggplot2::theme(axis.text =
+    ggplot2::element_text(size = 10), axis.title =
+    ggplot2::element_text(size = 10)) +
+    ggplot2::scale_x_continuous(expand = c(0.1, 0.2))
```

This code results in Figure 11. 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 11 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

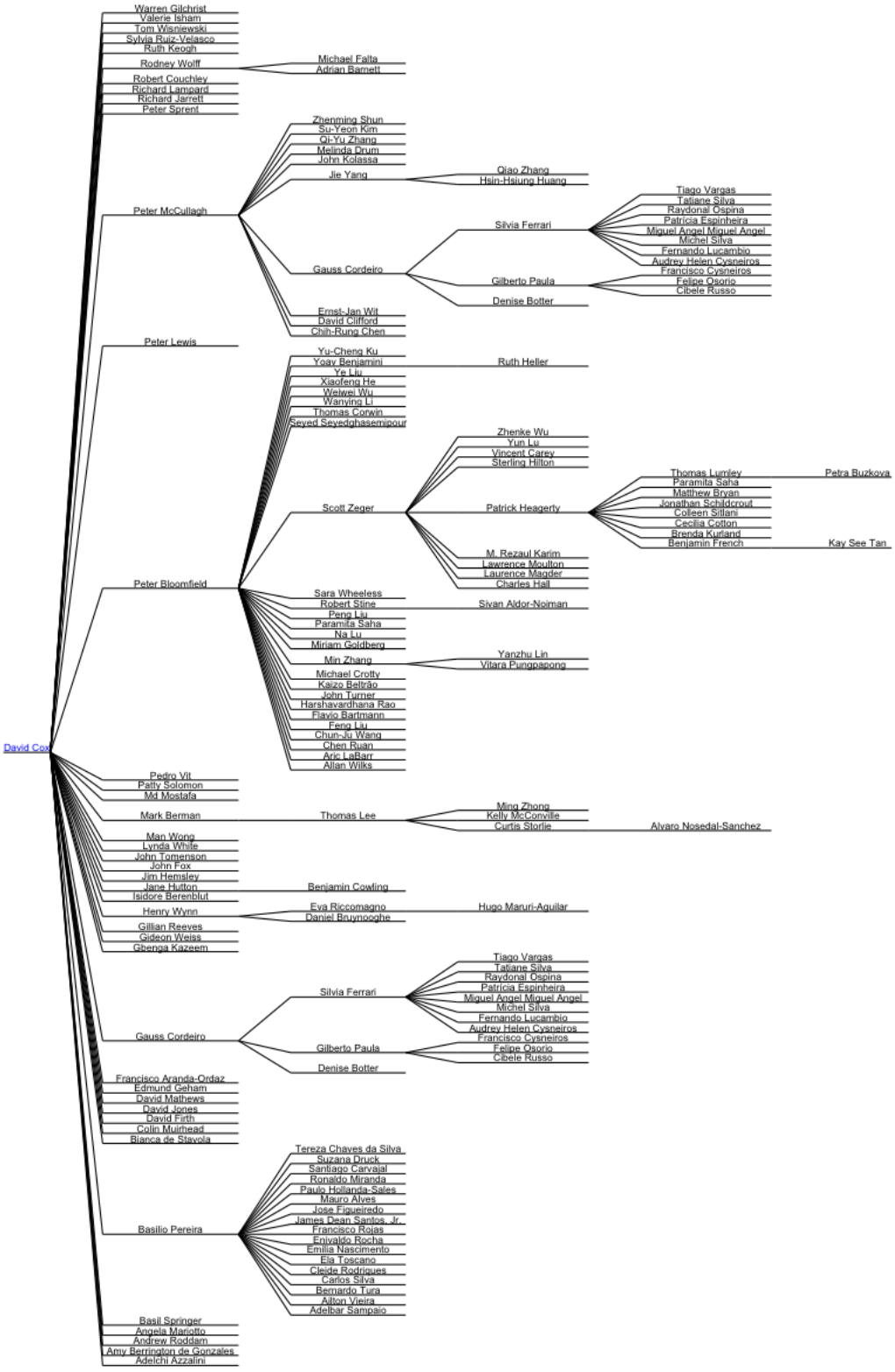

Figure 10: The 159 academic statistician "descendants" of Sir `David Cox`.

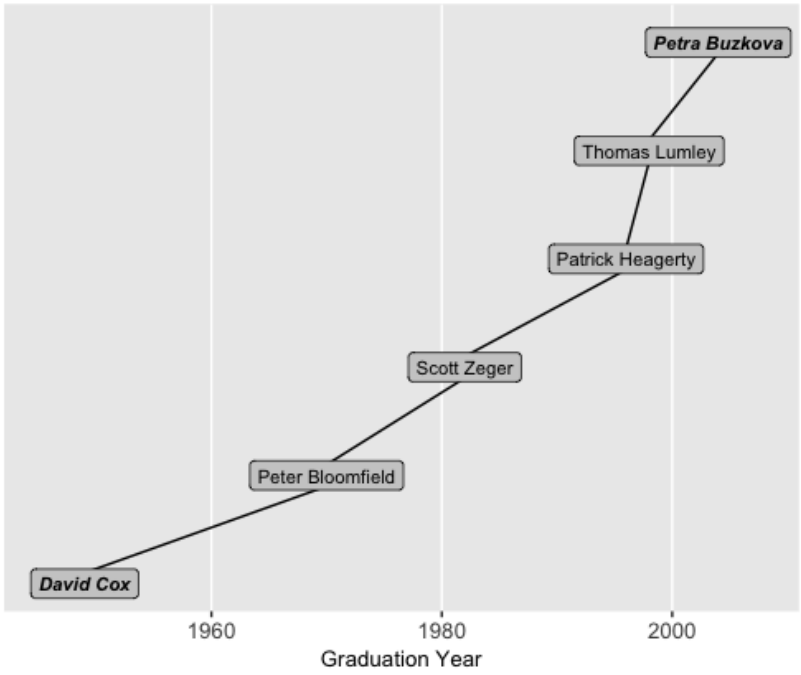

Figure 11: The shortest path between Sir `David Cox` and one of his fifth generation "descendants", `Petra Buzkova`.

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 5 and 6.

We can achieve this using the below code. After trial and error, we use a `bin` of size 200, and append **ggplot2** syntax to define suitable x-axis limits. The output of this process is illustrated in Figure 12.

```
R> plotPathOnAll(pathCB, statGeneal, statIG, "gradYear", bin = 200) +
+    ggplot2::theme(axis.text = ggplot2::element_text(size = 8),
+    axis.title = ggplot2::element_text(size = 8)) +
+    ggplot2::scale_x_continuous(expand = c(0.1, 0.2)) +
+    ggplot2::xlab("Graduation Year")
```

We see from the resulting Figure 12 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.

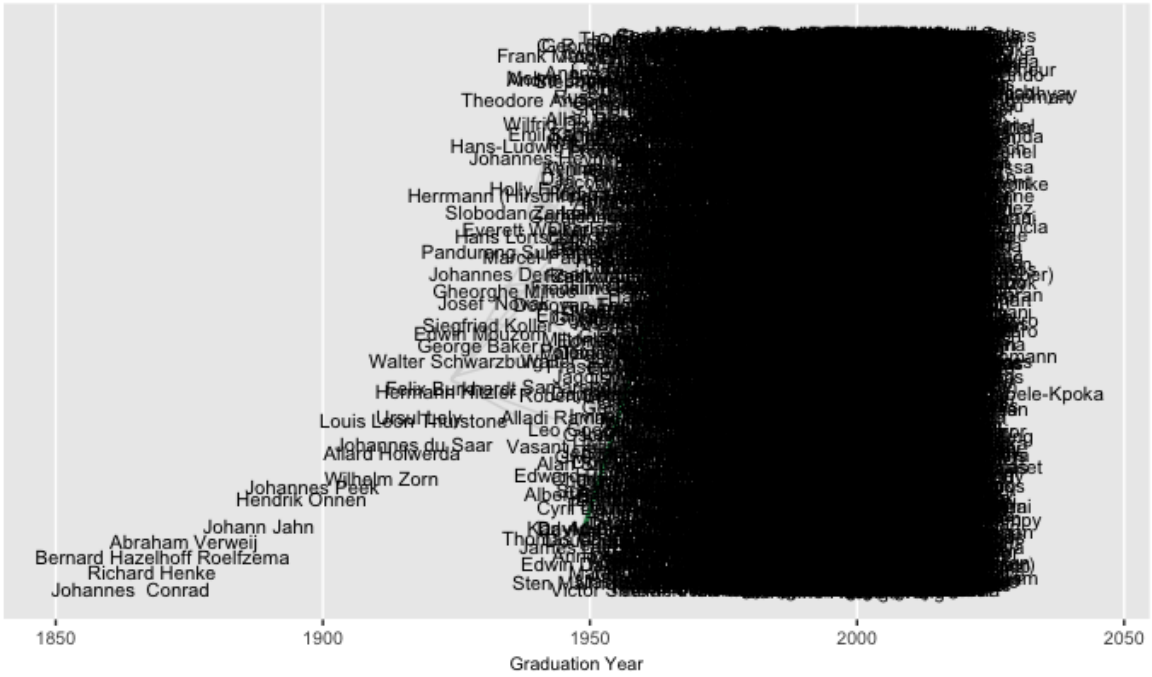

Figure 12: The shortest path between Sir `David Cox` and `Petra Buzkova`, superimposed over the data structure, using a bin size of 200.

The text labels are so numerous in Figure 12 that simply trying different values for the input parameter `bin` 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 that are 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 are 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 13.

```
R> plotPathOnAll(pathCB, statGeneal, statIG, "gradYear", bin = 200,
+    nodeSize = 0.5, 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(0.1, 0.2)) +
+    ggplot2::xlab("Graduation Year")
```

In Figure 13, 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.

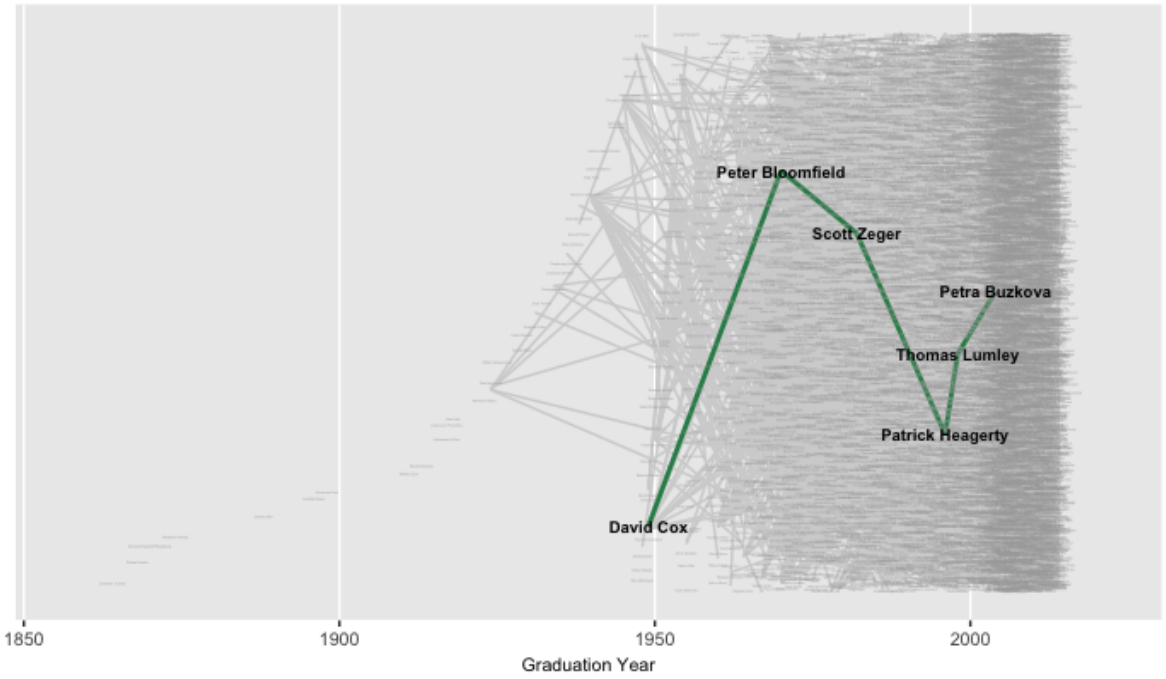

Figure 13: The shortest path between Sir `David Cox` and `Petra Buzkova`, superimposed over the data structure, using a bin size of 200. Individuals on the shortest path are labeled in large and black text and connected by dark green edges; all other individuals are labeled in small and gray text and connected by light gray edges.

12. Interactive plotting functions

We could still improve upon Figure 13. 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 13, 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. This interactive functionality comes from methods in the **plotly** package (Sievert *et al.* 2019). A short video demonstration of these interactive features can be viewed in Figure 14.

```
R> plotPathOnAll(pathCB, statGeneal, statIG, "gradYear", bin = 200,
+    nodeSize = 0.5, pathNodeSize = 2.5, nodeCol = "darkgray",
+    edgeCol = "lightgray", animate = TRUE)
```

We would like to state that users can still hard code one or two variables in function `plotPathOnAll()`, even with the `animate` option set to a value of `TRUE`. The first call to the `plotPathOnAll()` function below would produce an interactive plot with the yield on the

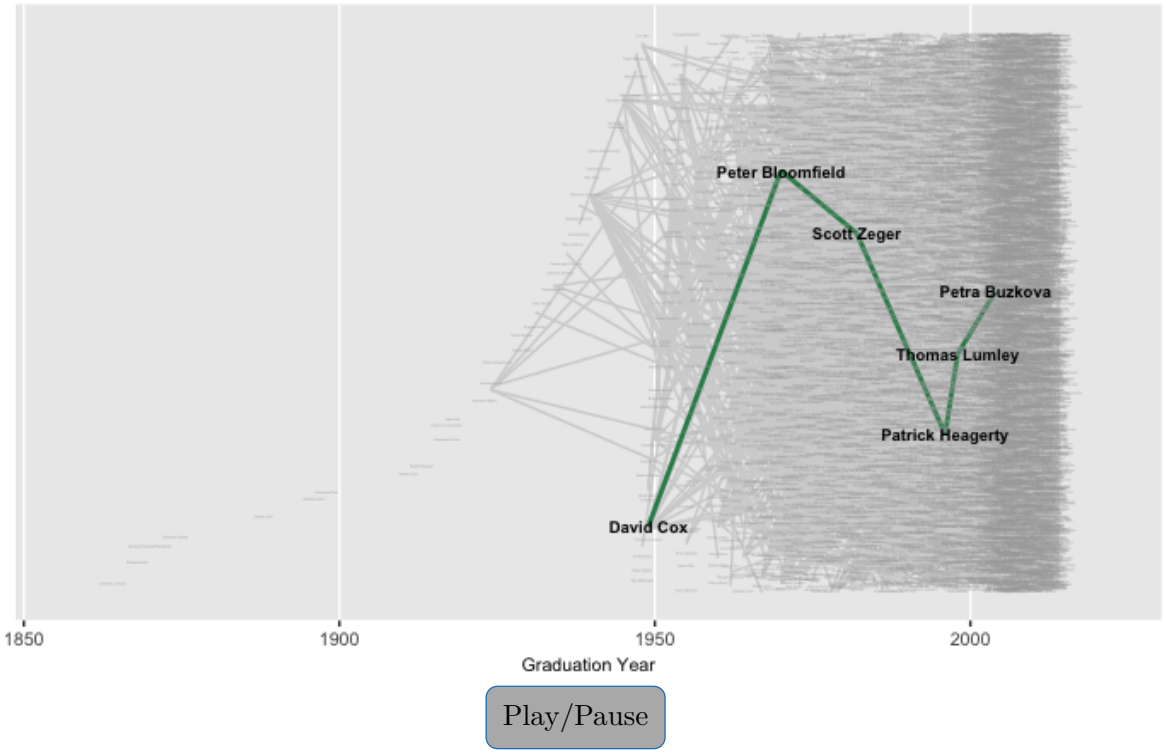

Figure 14: A short video demonstrating the animation features for this function. Please note that to properly view this video, the PDF version of this document must be opened in Adobe Acrobat Reader DC (version ≥ 9), which can be downloaded free of charge. This video can only be viewed on Windows or Mac OS X platforms; it cannot be viewed on mobile devices.

horizontal axis. When hovering over labels, the label name and yield value would be revealed. The second call to the `plotPathOnAll()` function below would produce an interactive plot with the yield on the horizontal axis and development year on the vertical axis. When hovering over labels, the label name, yield value, and development year would be revealed. We do not include in this document the animated videos that the below code creates, but readers can use the below code to create them on their own.

```
R> plotPathOnAll(pathCL, sbFilt, sbFiltIG, "yield", pathEdgeCol = "orange",
+    animate = TRUE)
R> plotPathOnAll(pathCL, sbFilt, sbFiltIG, "yield", "devYear",
+    pathEdgeCol = "orange", animate = TRUE)
```

13. Branch parsing and calculations

It may be helpful for users to search through descendant branches of a certain individual to compare and contrast how a variable of interest changes along those branches. For instance, which descending branches of a particular soybean variety are producing the highest yields? Which branches are developing new varieties in recent years? Which descending branches of a particular academic statistician have large proportions of students graduating from certain universities or countries? Which branches are graduating new students in recent years? Which branches have the highest proportion of thesis titles containing a word of interest?

Answering these questions in a straightforward manner requires more than basic data frame manipulation: It also requires methods that can easily traverse parent-child relationships. The **ggenealogy** package has two methods that can answer these questions using branch traversal. The `getBranchQuant()` function can be used to track a quantitative variable across branches and the `getBranchQual()` method can be used to track a qualitative variable across branches.

13.1. Quantitative variable parsing and calculations

We can demonstrate the `getBranchQuant()` function by examining the quantitative variable "yield" across the descendant branches of the soybean variety `A.K.` To understand more about the output of this function, please consult the **ggenealogy** package documentation. In the code below, we remove the output column "DesNames" because it verbosely lists all descendant names, which is not necessary for this demonstration.

```
R> AKBranchYield <- getBranchQuant("A.K.", sbGeneal, "yield", 15)
R> select(AKBranchYield, -DesNames)

           Name     Mean       SD Count NACount
1 A.K. (Harrow) 2932.154 197.0092    54      41
2        Illini 2856.667 210.7801   131     125
```

We see from the output that `A.K.` has two children named `A.K. (Harrow)` and `Illini`. Descendants from the `A.K. (Harrow)` branch have a higher mean yield than the `Illini` branch (2932.154 versus 2856.667). However, we should recognize that even though the branches contain a large number of descendants (54 and 131), most of these descendants did not come with a yield value (41 and 125). As a result, the mean values were calculated from a small proportion of the descendants.

As another example, we can examine the mean graduation year for the "descendant" branches of the academic statistician `David Cox`. We know from earlier that `David Cox` had 42 "children", so as expected, the `CoxBranchYear` object below contains 42 rows. However, only 8 of these rows have any "descendants" of their own. As a result, only the first 8 rows of the `CoxBranchYear` object contain branch information.

```
R> CoxBranchYear <- getBranchQuant("David Cox", statGeneal, "gradYear", 15)
R> head(select(CoxBranchYear, -DesNames), 10)

                         Name     Mean        SD Count NACount
1                Mark Berman 2007.200  6.340347     5       0
2                 Henry Wynn 2005.333  7.637626     3       0
3               Rodney Wolff 2003.500  2.121320     2       0
4                Jane Hutton 2003.000        NA     1       0
5              Gauss Cordeiro 2002.643  7.722167    14       0
6             Peter McCullagh 2001.231  8.778645    26       0
7             Basilio Pereira 2000.647 10.074356    17       0
8           Peter Bloomfield 1999.918 11.707969    49       0
9           Adelchi Azzalini      NaN        NA     0       0
10 Amy Berrington de Gonzales      NaN        NA     0       0
```

In this case, we see that of the 8 "children" of `David Cox` who had "children" of their own, `Mark Berman` had the "descendants" ($n = 5$) who have on average graduated the most recently (2007.200), whereas `Peter Bloomfield` has the "descendants" ($n = 49$) who on average have graduated the least recently (1999.918). We see that, for all branches, there are no "descendants" who contain an `NA` value for graduation year.

13.2. Qualitative variable parsing and calculations

The `getBranchQual()` function requires similar inputs as the `getBranchQuant()` function above, except that it also requires an input parameter called `rExpr`. The user must initialize this input parameter to a regular expression that can be applied to the column containing the qualitative variable of interest. The regular expression syntax must work on a data frame column of type character. It must be saved as a double quotation string, and any quotation marks within it must be single quotations. The term `geneal$colName` must be used in the regular expression.

We can demonstrate the `getBranchQual()` function by examining the qualitative variable "thesis" across the "descendant" branches of the academic statistician `David Cox`. Since one of the primary research areas for `David Cox` was stochastic processes, we can determine if any descendant branches of his "children" contained thesis titles that included the word "stochastic".

```
R> v1 <- "David Cox"; geneal <- statGeneal; colName <- "thesis"; gen <- 15
R> rExpr <- "grepl('(?i)Stochastic', geneal$colName)"
R> CoxBranchStochastic <- getBranchQual(v1, geneal, colName, rExpr, gen)
R> head(select(CoxBranchStochastic, -DesNames))
```

```
                        Name CountTrue Count NACount
1           Peter Bloomfield         4    49       0
2            Basilio Pereira         1    17       0
3           Adelchi Azzalini         0     0       0
4 Amy Berrington de Gonzales         0     0       0
5              Andrew Roddam         0     0       0
6            Angela Mariotto         0     0       0
```

We see that only two "children" of `David Cox` had any "descendants" with thesis titles containing the word "Stochastic" (4 out of 49 "descendants" of `Peter Bloomfield` and 1 out of 17 "descendants" of `Basilio Pereira`). We see again that none of the "descendants" from either branches contained values that were `NA` for the variable "thesis".

In many string parsing applications, the choice of the regular expression can be tricky. This is true when the string variable we are parsing contains thesis titles. For instance, notice that in our regular expression, we accounted for all instances of the substring "Stochastic". Hence, words that contain "Stochastic" (such as "Stochastics" and "Stochastically") will also be returned. In addition, we defined our regular expression to return matches whether the first letter was upper or lower case. When initializing the `rExpr` parameter, users would need to consider what nuances of their search criteria they would like to define as matches.

We will demonstrate one more example of the `getBranchQual()` function by searching the qualitative variable "school" across the "descendant" branches of the academic statistician

David Cox. The Mathematics Genealogy Project coding system for the "school" variable was non-ambiguous, and so we do not have to worry about all the various ways the same school could be coded in the dataset. As a result, we no longer have to search for various substrings; we can simply use a regular expression that equates to one value.

It may be interesting to examine the school that is represented the most among all descendants of David Cox. To determine what school this is, we use the getDescendants() function to create a data frame called desDC that contains the names of all 159 "descendants" of David Cox. Then, we use the base R function match() to match the school names from the original genealogy dataset to each of the 159 "descendants" in the desDC data frame. After that, we use the base R functions sort() and table() to examine the five schools that were represented the most throughout the 159 "descendants".

```
R> desDC <- getDescendants("David Cox", statGeneal, 15)
R> tableDC <- table(statGeneal[match(desDC$label,
+    statGeneal$child), ]$school)
R> tail(sort(tableDC), 5)
```

```
     The Johns Hopkins University  Universidade Federal do Rio de Janeiro
                                9                                       17
  North Carolina State University              Universidade de São Paulo
                               18                                       28
            University of London
                               35
```

We see from this table that the most common school of the 159 "descendants" of David Cox was the University of London with a count of 35. We can now determine which of the branches from the 42 "children" of David Cox have the largest proportion of "descendants" graduating from the University of London.

```
R> colName <- "school"
R> rExpr <- "geneal$colName == 'University of London'"
R> DCBranchUL <- getBranchQual(v1, geneal, colName, rExpr, gen)
R> head(select(DCBranchUL, -DesNames))
```

```
                       Name CountTrue Count NACount
1            Peter McCullagh         1    26       0
2           Adelchi Azzalini         0     0       0
3 Amy Berrington de Gonzales         0     0       0
4              Andrew Roddam         0     0       0
5            Angela Mariotto         0     0       0
6             Basil Springer         0     0       0
```

We see that Peter McCullagh is the only "child" of David Cox that has a "descendant" branch with one student graduating from the University of London; the rest of the 41 children of David Cox have "descendant" branches with zero students graduating from the University of London. This must mean the other 34 "descendants" of David Cox that graduated from the University of London were direct "children" of David Cox. We can verify this below:

```
R> DCChild <- statGeneal[match(getChild("David Cox", statGeneal),
+    statGeneal$child), ]
R> sum(DCChild$school == "University of London")

[1] 34
```

These examples demonstrate that users can quickly and flexibly parse descendant branches. The swiftness comes from **ggenealogy** functions that allow for fast parent-child traversals, such as `getChild()`, `getDescendants()`, `getBranchQuant()`, and `getBranchQual()`. The flexibility comes from data frame manipulation functions in base R that can be used in conjunction with the parent-child traversal methods.

14. Future avenues

Incorporation of the **shiny** application allows users to examine **ggenealogy** tools in a more interactive way (Chang, Cheng, Allaire, Xie, and McPherson 2019). The reactive programming saves them the time of using the 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 datasets. 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 6). 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 6. If we could subsequently fine-tune the vertical positions of the small fraction of text labels that remained overlapped after application of 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 12). 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 13).

15. 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.

Acknowledgments

The authors thank Drs. James E. Specht and Randy C. Shoemaker for helpful discussions of soybean genealogy. In addition, the authors 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. Author Cook was a faculty member at Iowa State University at the time that most of this work was conducted.

References

Chang W, Cheng J, Allaire JJ, Xie Y, McPherson J (2019). ***shiny**: Web Application Framework for R*. R package version 1.3.2, URL https://CRAN.R-project.org/package=shiny.

Coster A (2013). ***pedigree**: Pedigree Functions*. R package version 1.4, URL https://CRAN.R-project.org/package=pedigree.

Csardi G, Nepusz T (2006). "The **igraph** Software Package for Complex Network Research." *InterJournal, Complex Systems*, **1695**.

Gansner ER, North SC (2000). "An Open Graph Visualization System and Its Applications to Software Engineering." *Software: Practice and Experience*, **30**(11), 1203–1233. doi:10.1002/1097-024x(200009)30:11<1203::aid-spe338>3.3.co;2-e.

Hymowitz T, Newell CA, Carmer SG (1977). *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.

North Dakota State University, American Mathematical Society (2010). *The Mathematics Genealogy Project*. Archived Web Site. Retrieved from the Library of Congress, Accessed on 2015-03-06, URL http://www.genealogy.math.ndsu.nodak.edu.

PostgreSQL (2016). "PostgreSQL: The World's Most Advanced Open Source Relational Database." URL http://www.postgresql.org/.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rutter L, Vanderplas S, Cook D (2019). ***ggenealogy**: Visualization Tools for Genealogical Data*. R package version 1.0.0, URL https://CRAN.R-project.org/package=ggenealogy.

Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B, Ideker T (2003). "**Cytoscape**: A Software Environment for Integrated Models of Biomolecular Interaction Networks." *Genome Research*, **13**(11), 2498–2504. doi:10.1101/gr.1239303.

Sievert C, Parmer C, Hocking T, Chamberlain S, Ram K, Corvellec M, Despouy P (2019). ***plotly**: Create Interactive Web Graphics via* `plotly.js`. R package version 4.9.0, URL https://CRAN.R-project.org/package=plotly.

Therneau T, Daniel S, Sinnwell J, Atkinson E (2015). ***kinship2**: Pedigree Functions*. R package version 1.6.4, URL https://CRAN.R-project.org/package=kinship2.

Wickham H (2009). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer-Verlag, New York. doi:10.1007/978-0-387-98141-3.

Wickham H, François R, Henry L, Müller K (2019). ***dplyr**: A Grammar of Data Manipulation*. R package version 0.8.1, URL https://CRAN.R-project.org/package=dplyr.

Affiliation:

Lindsay Rutter
Bioinformatics and Computational Biology Program
Iowa State University
2014 Molecular Biology Building
Ames, IA, 50011, United States of America
E-mail: lrutter@iastate.edu
URL: https://github.com/lindsayrutter/

Susan VanderPlas
Department of Statistics
Iowa State University
2413 Snedecor Hall
Ames, IA, 50011, United States of America
E-mail: srvanderplas@gmail.com
URL: https://github.com/srvanderplas/

Dianne Cook
Department of Econometrics and Business Statistics
Monash University
E869 Menzies Building
20 Chancellors Walk
Clayton, VIC 3800, Australia
E-mail: dicook@monash.edu
URL: https://github.com/dicook/

Michelle Graham
USDA Agriculture Research Service, Corn Insects and Crop Genetics Research Unit
Department of Agronomy, Iowa State University
1565 Agronomy Building
Ames, IA, 50011, United States of America
E-mail: michelle.graham@ars.usda.gov

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
May 2019, Volume 89, Issue 13 *Submitted:* 2016-05-27
doi:10.18637/jss.v089.i13 *Accepted:* 2017-10-17