Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. doi: 10.18637/jss.v000.i00

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
Iowa State University

Abstract

This paper introduces **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. 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 *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 is currently available on the Comprehensive R Archive Network.

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** 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. 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 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 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.

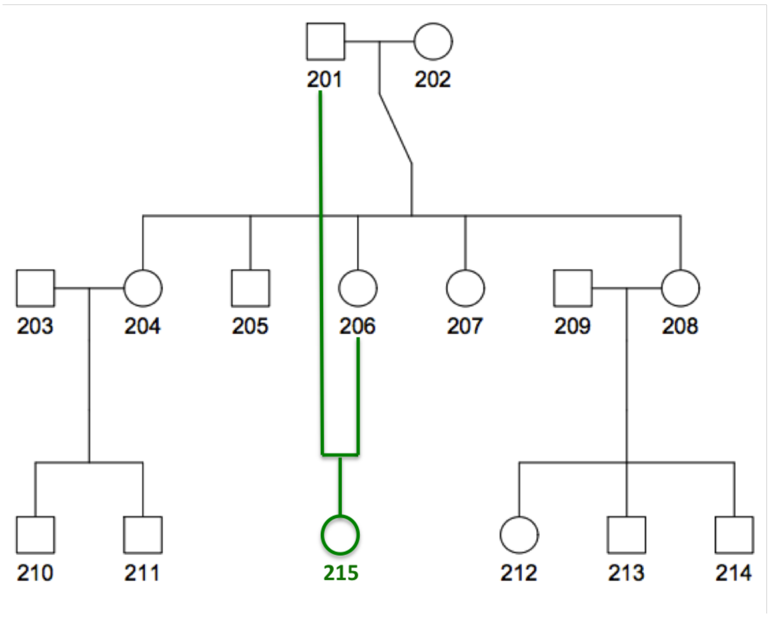

Figure 1: Example pedigree chart from the kinship2 package, where the vertical axis denotes generation count. We superimposed green-highlighted individual 215 for explanatory purposes. As an offspring of a parent-child relationship, individual 215 is both a second and third generation individual. Hence, 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 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 8, we will demonstrate **ggenealogy** plots that can remedy these problems.

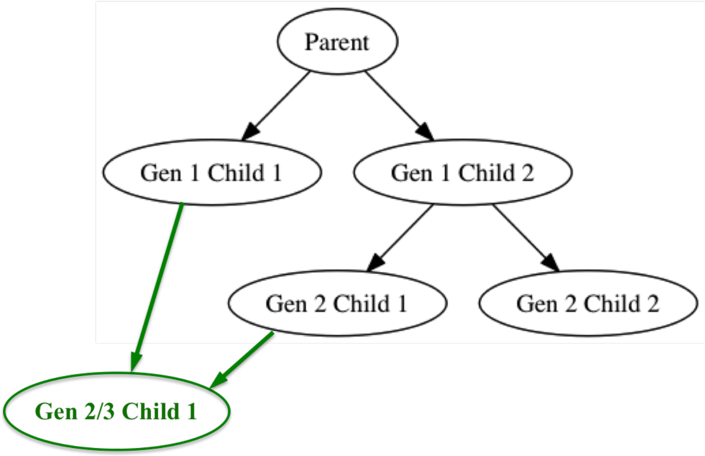

Figure 2: Example genealogical display using popular graph software like GraphViz and Cytoscape, with generation count denoted by the vertical axis. As was shown in Figure 1, the green node has parents from two different generations, and hence must be ambiguously positioned as one of two generation counts.

3. 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.

3.1. 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." ...
```

3.2. 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, 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"
```

4. 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.

5. 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
```

6. 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, fontFace = 2)
```

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 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)
```

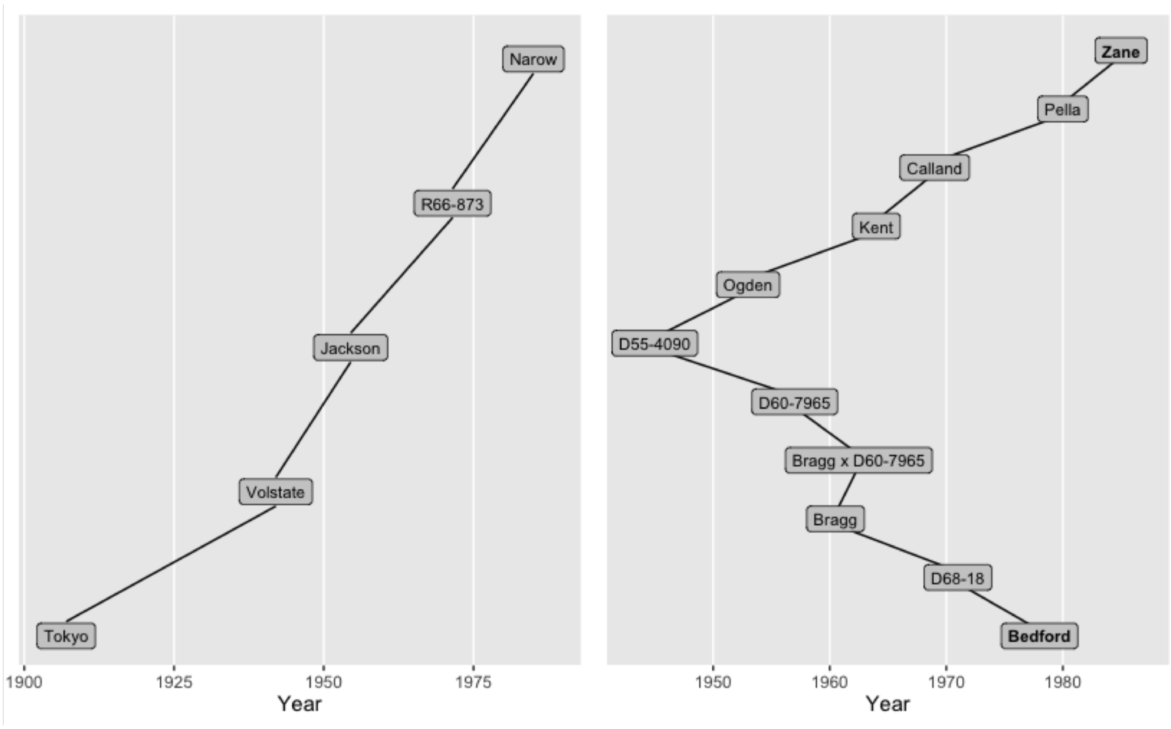

Figure 3: Left: The shortest path between varieties Tokyo and Narow is strictly composed of a unidirectional sequence of parent-child relationships. Right: The shortest path between varieties Zane and Bedford is not strictly composed of unidirectional parent-child relationships; they instead have a cousin-like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

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 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.

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

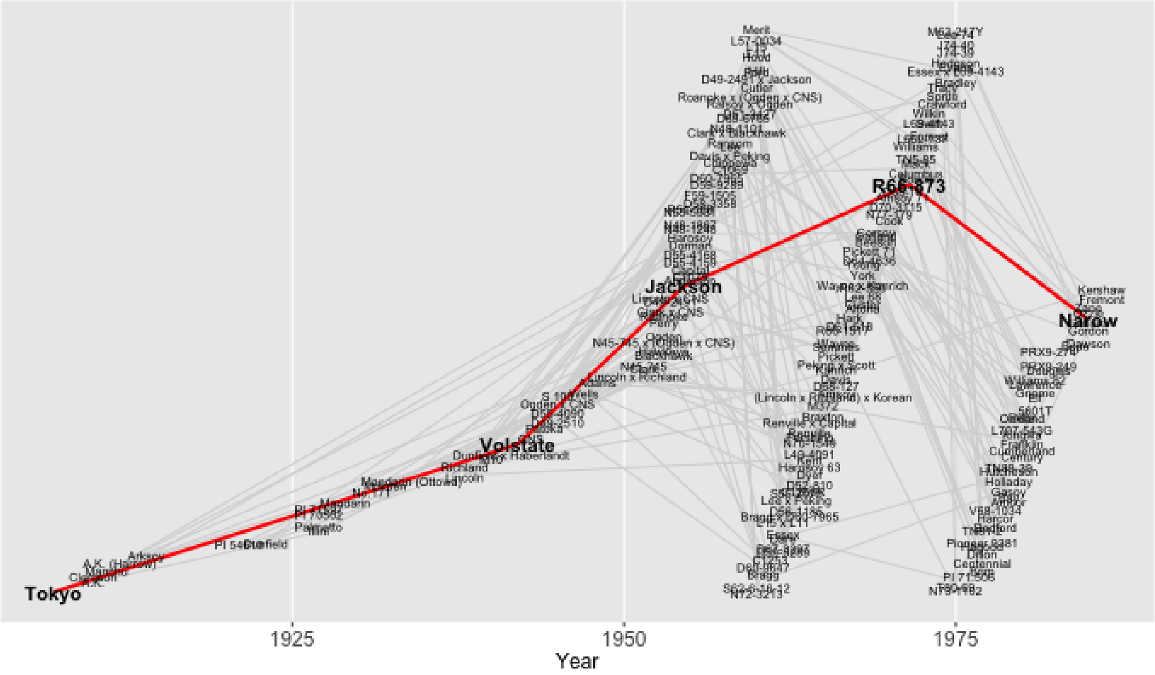

Figure 4: The shortest path between Tokyo and Narow, superimposed over the data structure, using a bin size of 3.

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 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 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 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 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 5.

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

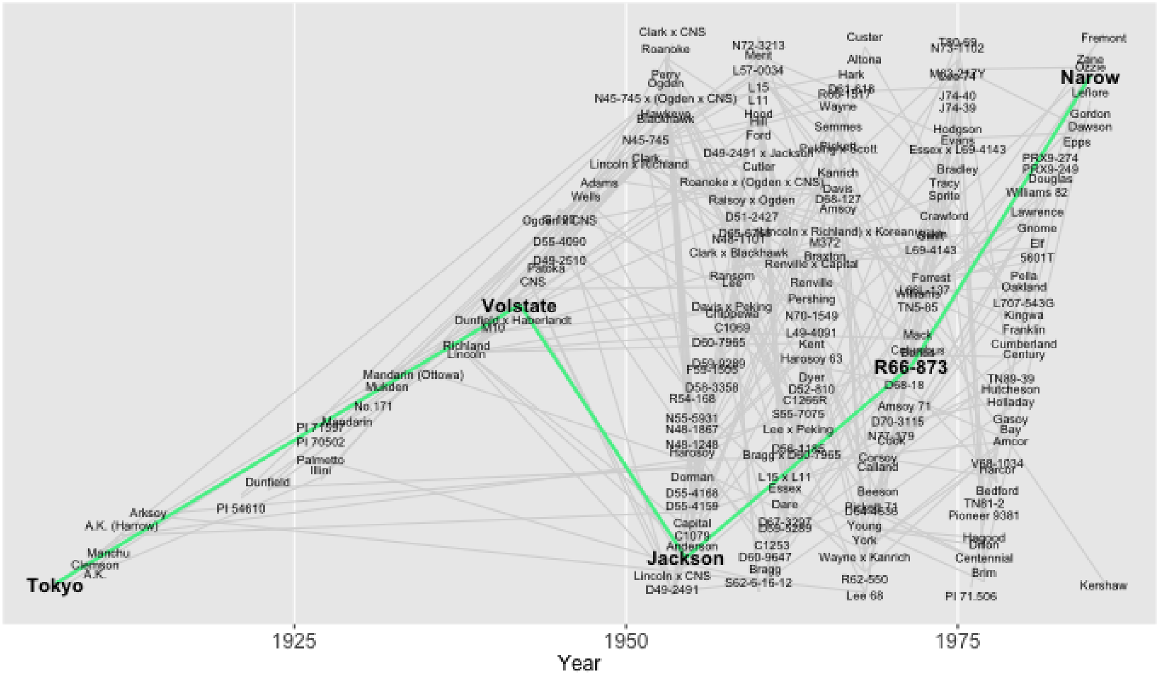

Figure 5: The shortest path between Tokyo and Narow, superimposed over the data structure, using a bin size of 6.

We can immediately see that Figure 5 more successfully mitigates text overlap compared to Figure 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.

8. 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 1 and 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 plot in the left side of Figure 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.

The right side of Figure 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 left 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 right as it can be in the **ggenealogy** graph on the left. This is a current limitation in other genealogy and graphical software that **ggenealogy** can now provide.

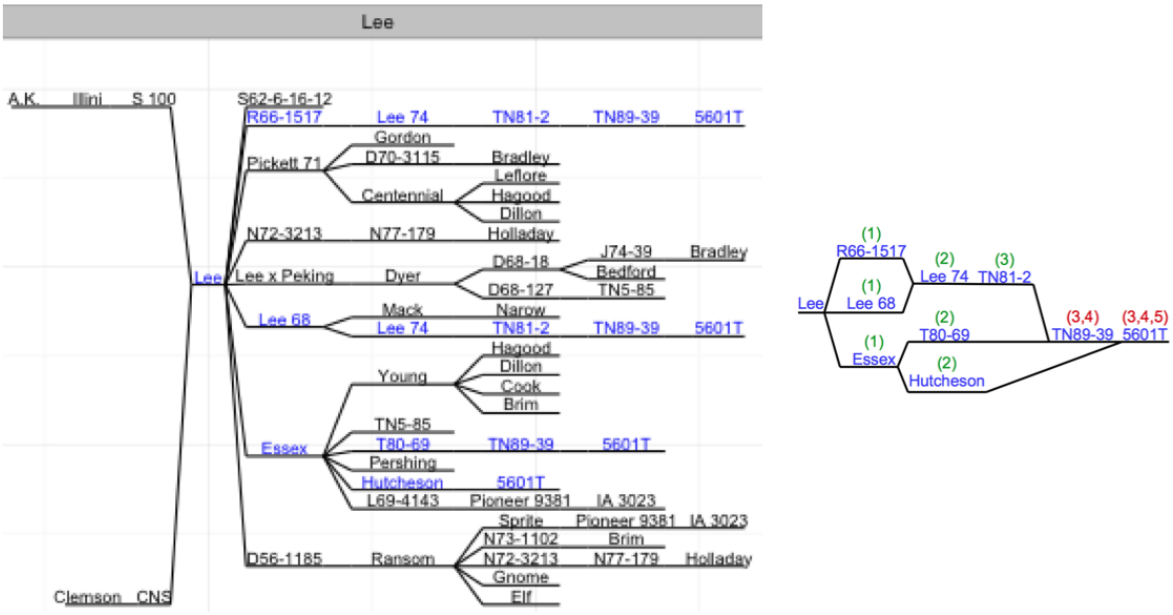

Figure 6: Left: All ancestors and descendants of the variety Lee are shown in this ggenealogy plot. Right: We now attempt to mimic the blue paths in the ggenealogy plot on the left, only now nodes cannot be repeated. The parenthetical numbers above each node represents the set of generation counts that node is away from the center node Lee. The presence of red parentheses indicate that the plot on the right ambiguously display the example soybean genealogy in the way that the ggenealogy plot on the left can accomplish.

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

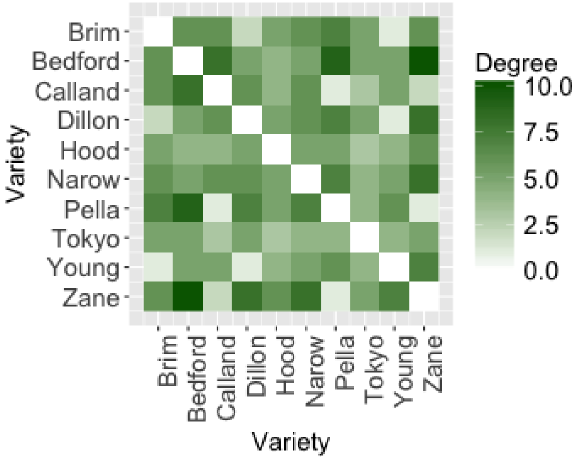

Figure 7: The shortest path degree matrix between ten varieties of interest.

10. 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 3, 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 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 6. As such, we create Figure 8 using the code below.

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

We see from Figure 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.

```
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.

```
R> statIG <- dfToIG(statGeneal)
R> pathCB <- getPath("David Cox", "Petra Buzkova", statIG, statGeneal,
  isDirected = FALSE)
R> plotPath(pathCB) + ggplot2::theme(axis.text = ggplot2::element_text(size
  = 12), axis.title = ggplot2::element_text(size = 12)) +
  ggplot2::scale_x_continuous(breaks = seq(1940, 2010, by = 10))
```

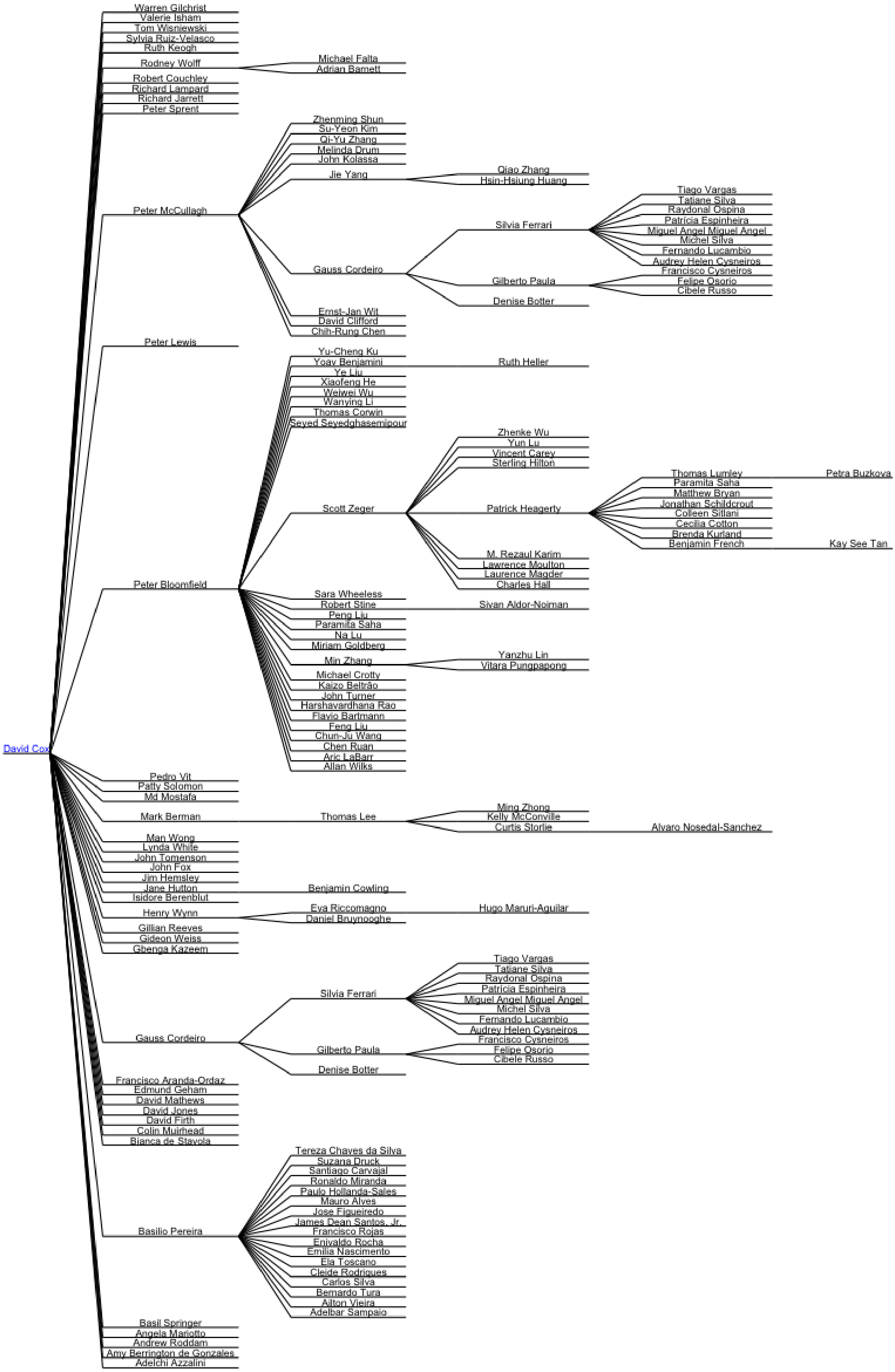

Figure 8: The 159 academic statistician “descendants” of Sir David Cox.

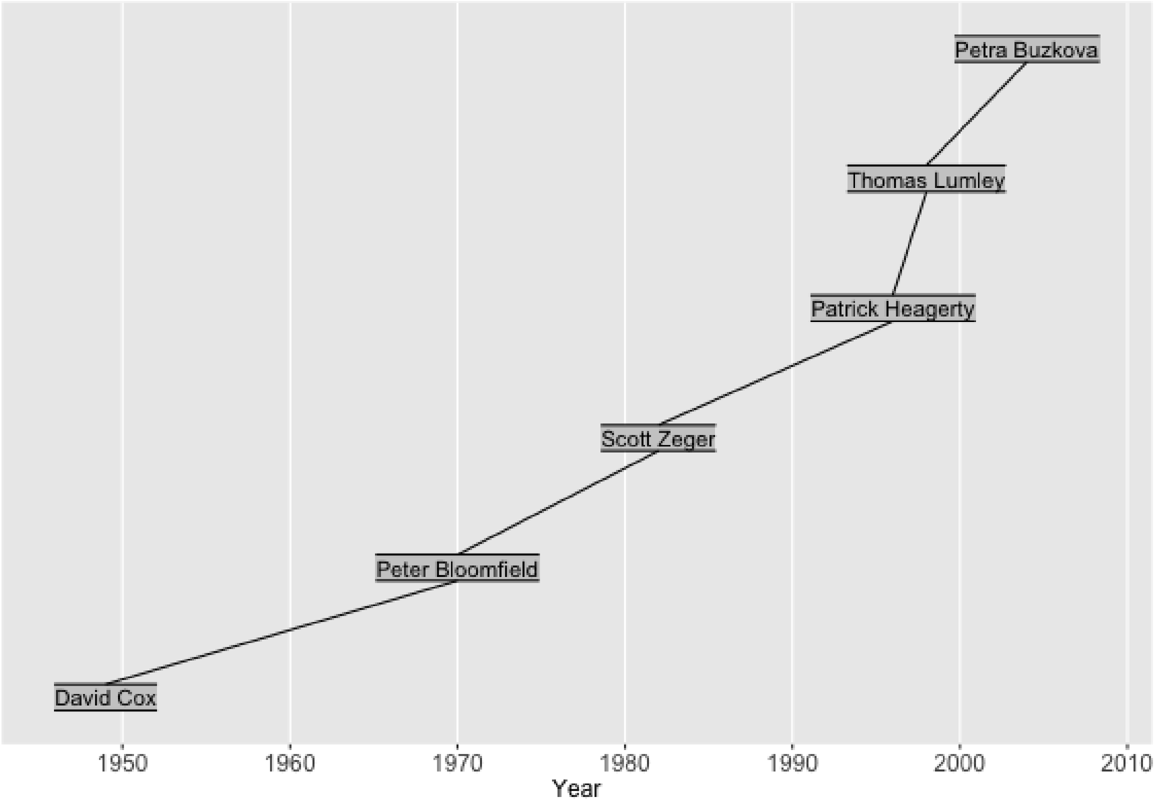

Figure 9: The shortest path between Sir David Cox and one of his fifth generation "descendants", Petra Buzkova.

This code results in Figure 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 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 4 and 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 10.

```
R> plotPathOnAll(pathCB, statGeneal, statIG, binVector = 1:200) + ggplot2::
  scale_x_continuous(limits = c(1850, 2030)) + ggplot2::theme(axis.text =
  ggplot2::element_text(size = 12), axis.title = ggplot2::element_text(size
  = 12))
```

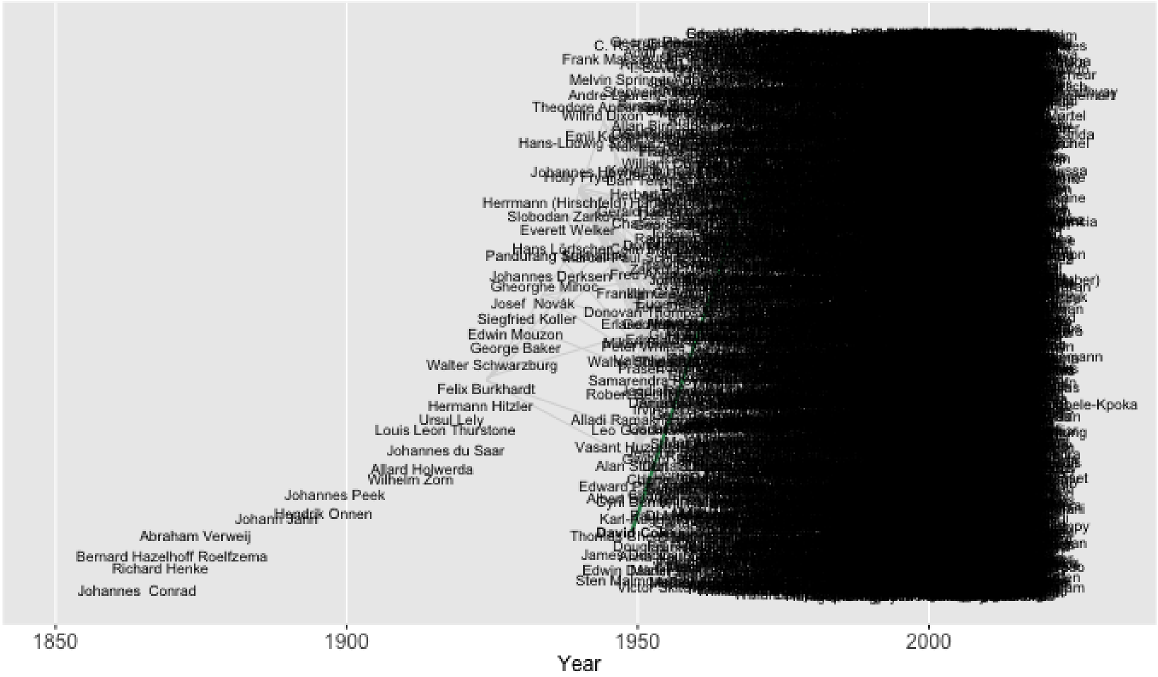

Figure 10: The shortest path between Sir David Cox and Petra Buzkova, superimposed over the data structure, using a bin size of 200.

We see from the resulting Figure 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 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 a value of `TRUE` for the input parameter `nodeLabel`. As is further detailed in the help manual of the function, doing so will generate a plot where only text labels for individuals on the path of interest are labeled; all other individual labels are represented as dots. We provide the code for this below, which results in Figure 11.

```
R> plotPathOnAll(pathCB, statGeneal, statIG, binVector = 1:200, nodeSize = 4,
  pathNodeSize = 4, nodeLabel=FALSE) + ggplot2::theme(axis.text = ggplot2::
  element_text(size = 12), axis.title = ggplot2::element_text(size = 12)) +
  ggplot2::xlab(1850, 2025)
```

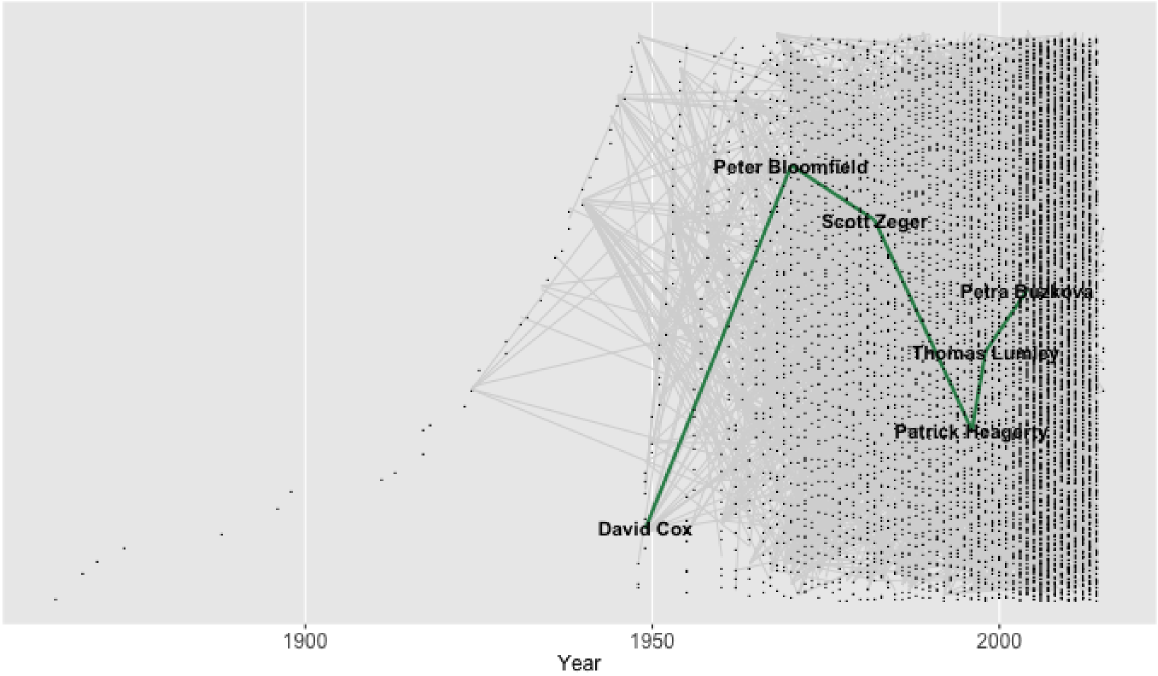

Figure 11: The shortest path between Sir David Cox and Petra Buzkova, superimposed over the data structure, using a bin size of 200. Individuals on the shortest path have text labels; all other individuals have dot labels.

In Figure 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.

11. 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 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 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 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 11).

12. 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

Coster A (2013). *pedigree: Pedigree functions.* R package version 0.4, 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**. URL `http://igraph.sf.net`.

Gansner ER, North SC (2000). "An open graph visualization system and its applications to software engineering." *Software - Practice and Experience*, **30**(11), 1203–1233.

Hymowitz T, Newell C, Carmer S (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 and American Mathematical Society (2010). *The Mathematics Genealogy Project*. Archived Web Site. Retrieved from the Library of Congress, Accessed on March 6, 2015, URL `http://www.genealogy.math.ndsu.nodak.edu`.

PostgreSQL (2016). URL `http://www.postgresql.org/`.

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

Rutter L, Vanderplas S, Cook D (2015). *ggenealogy: Visualization Tools for Genealogical Data*. R package version 0.1.0, 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.

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, NY. URL `http://had.co.nz/ggplot2/book`.

Wickham H, Francois R (2015). *dplyr: A Grammar of Data Manipulation*. R package version 0.4.3, 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/lrutter/`

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/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd