Replication Script to JSS Submission

Lindsay Rutter, Susan VanderPlas, Dianne Cook, Michelle A. Graham

May 21, 2016

2. Available software

The two figures in this section of the paper (Figure 1 and 2) were not produced using source code from our ggenealogy package. Instead, they were produced from external software for demonstration purposes.

3. Example datasets

First, we can load and examine the structure of the example soybean genealogy dataset (called sbGeneal):

```
> rm(list=)
> library("devtools")
> load_all("ggenealogy")
> library("ggenealogy")
> data("sbGeneal")
> 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." ...
```

After that, we can load and examine the structure of the example academic statistician genealogy dataset (called statGeneal):

```
> data("statGeneal")
> dim(statGeneal)

[1] 8165    6

> colnames(statGeneal)

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

5. Generating a graphical object

We can convert our example soybean genealogy dataset (called sbGeneal) into an igraph object:

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

Now that we have an igraph object, we can obtain basic summary statistics about it:

```
> 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

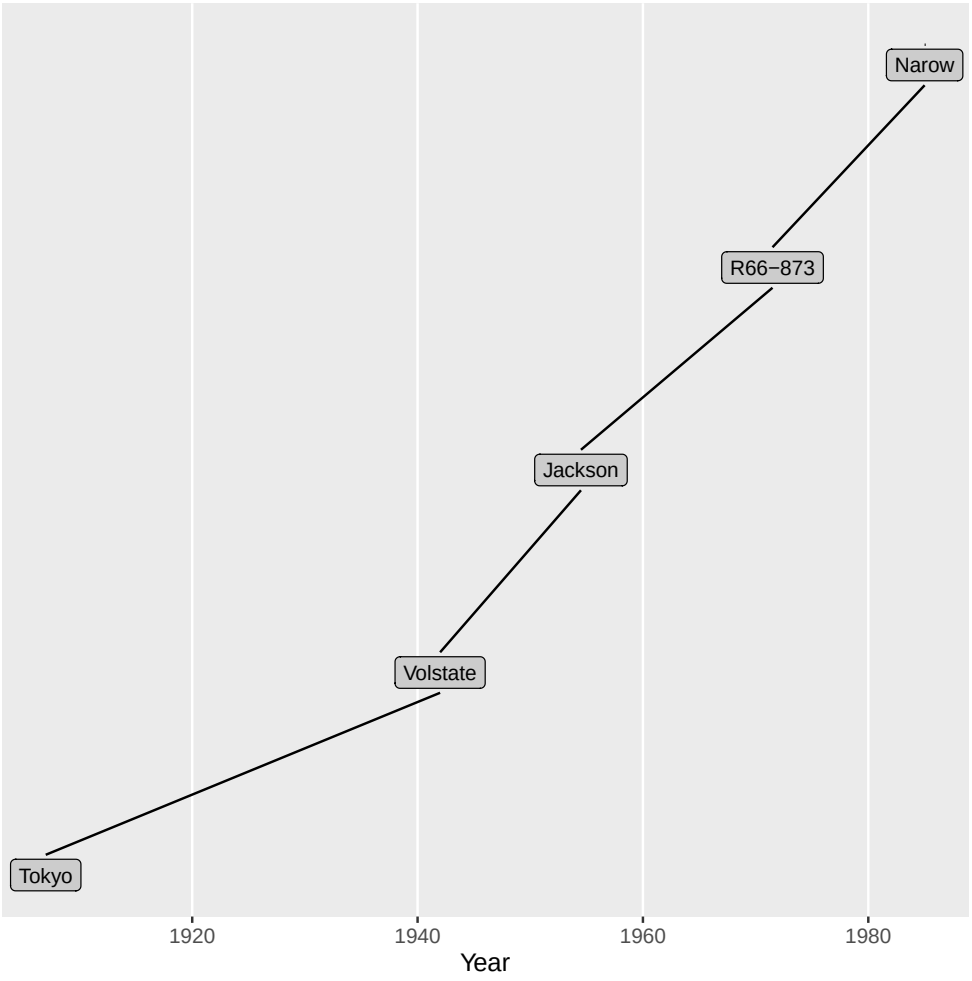
At this point, we can determine the shortest path of parent-child relationships between two labels of interest (”Tokyo” and ”Narow”).

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

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

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

The returned path can then be plotted.

```
> plotPath(pathTN)
```

```
> plotPath(pathTN)
```



We can do the same for a different pair of two labels of interest (”Bedford” and ”Zane”). First, we can determine the years these two labels were identified.

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

[1] 1978

> getYear("Zane", sbGeneal)

[1] 1985
```

Next, we can determine the shortest path of parent-child relationships between these two labels of interest and plot it.

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

7. Superimposing shortest path on tree

In the previous section, we obtained the shortest path between the the pair of labels ”Tokyo” and ”Narow” and saved it as a variable pathTN. Here, we can plot that path superimposed over all labels in the example soybean genealogy dataset.

```
> plotPathOnAll(pathTN, sbGeneal, sbIG, binVector = 1:3, pathEdgeCol = "red", nodeSize = 2.5, pathNodeSi
```

We can repeat this process, only now instead of setting the binVector variable to 1:3 (as we did earlier), we can set it to 1:6.

```
> plotPathOnAll(pathTN, sbGeneal, sbIG, binVector = 1:6, pathEdgeCol = "seagreen2", nodeSize = 1, pathNo
```

8. Plotting ancestors and descendants by generation

As is explained in the article, only the top part of Figure 6 (the figure from this section) is produced by ggenealogy code. In contrast, the bottom part of Figure 6 was produced by tools outside of ggenealogy for didactic purposes. Below, we recreate the top part of Figure 6, which was to generate a plot of the ancestors and descendants of the label Lee.

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

9. Plotting distance matrix

We can plot the distance matrix for a set of 10 varieties.

```
> varieties <- c("Brim", "Bedford", "Calland", "Dillon", "Hood", "Narow", "Pella", "Tokyo", "Young", "Za
> plotDegMatrix(varieties, sbIG, sbGeneal, "Variety", "Variety", "Degree") + ggplot2::scale_fill_continu
```

10. Academic genealogy of statisticians

We can now explore some of the plotting functions in ggenealogy, only now with the academic statistican genealogy dataset. This second example dataset is much larger than the first example dataset of soybean genealogy. For example purposes, we would like to view the ancestor and descendant plot for the individual who has the largest number of descendants. To identify the name of this individual, we run the following code:

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

numDesc
   0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
6251  322  145   88   58   36   31   22   23   14   17   13   14   10    9    6
  16   17   18   19   20   21   22   23   24   25   26   27   29   30   34   37
   4    3    2    5    7    4    3    3    2    2    6    1    1    3    2    1
  38   40   41   44   45   48   49   60   61   75   77   84  159
   1    1    1    1    1    1    2    1    1    1    1    1    1

> which(numDesc == 159)

David Cox
     1980
```

We see the individual with the largest number of descendants is Sir David Cox, who has 159 descendants. Now, we can plot the ”ancestors” and ”descendants” of Sir David Cox.

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

It seems that of the 42 "children" of Sir David Cox, the one who went on to have the largest number of "children" of his own was Peter Bloomfield. We can verify below that Peter Bloomfield had 26 "children" and 49 "descendants".

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

```
[1] 26
```

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

```
[1] 49
```

It would be of interest now to examine the shortest path between Sir David Cox and one of his newest "descendants" Petra Buzkova. To do so, we first need to obtain the corresponding igraph object of the example academic statistican genealogy dataset.

```
> statIG <- dfToIG(statGeneal)
```

After doing so, we can now determine the shortest path between Sir David Cox and Petra Buzkova, and plot it.

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

We can now superimpose this shortest path between Sir David Cox and Petra Buzkova across the entire genealogical structure.

```
> plotPathOnAll(pathCB, statGeneal, statIG, binVector = 1:200) + ggplot2::theme(axis.text = ggplot2::ele
```

We notice, however, that we cannot read the text of the nodes on the path of interest. To solve this problem, we can create the same plot, only now specifying that any nodes that are not on our path of interest are deemphasized with smaller text.

```
> plotPathOnAll(pathCB, statGeneal, statIG, binVector = 1:200, nodeSize = .5, pathNodeSize = 2.5, nodeCo
```

Even though we can now read the text labels on the path of interest, we lost the ability to read labels that are not on our path of interest. At this point, we can keep the best of both worlds: We can create a plot that demphasizes the nodes that are not on the path of interest by assigning them small text font size, but incoroporate an interactive function so that we can hover over these non-path nodes if we wish to obtain their label information. This is the plot we used and interacted with to create the video embedded in Figure 12.

```
> plotPathOnAll(pathCB, statGeneal, statIG, binVector = 1:200, nodeSize = .5, pathNodeSize = 2.5, nodeCo
```