Chapter 10 examples

Brooke Anderson

4/22/2020

```r
library(tidyverse)
```

For Chapter 10, we're focusing on section 10.4.

The data comes from a package called `phangorn`. Make sure you install and load it first:

```r
# install.packages("phangorn")
library("phangorn")
```

This package has a nice vignette on estimating phylogenetic trees, available at: https://cran.r-project.org/web/packages/phangorn/vignettes/Trees.pdf. The package also has several other vignettes, which would be worth looking at if you did a project involving these methods.

The example code also uses functions from the package `ggtrees`, which is a ggplot extension for plotting trees. Make sure you load and install that package, as well (it's on Bioconductor, so use `BiocManager::install`):

```r
# BiocManager::install("ggtree")
library(ggtree)
```

The vignette for this package is actually a whole online book, available at: https://yulab-smu.github.io/treedata-book/.

The data for this section is from the book's data. It's in a file that ends with ".RData", which means it's in a binary R format. If you try to open it with a text editor, you won't be able to understand much (unlike a flat file format like CSV or TSV), but its binary format allows it to take up a bit less file space, which is helpful when you're working with large biological files.

The `load` function in R lets you load in files in this ".RData" format. Therefore, if you downloaded that data and copied it into your current project, then you can read it in with something like (the exact filepath will depend on how you've structured your directory):

```r
load("data/tree1.RData")
```

You should now see that you have ab object in your workspace named "tree" (the filename of the data file):

```r
ls()
```

```
## [1] "tree1"
```

The `load` command has loaded the data and created an object with the filename. You can check this out with `str` and `class`:

```r
tree1 %>%
  class()
```

```
## [1] "phylo"
```

```r
tree1 %>%
  str()
```

```
## List of 4
##  $ edge       : int [1:10, 1:2] 7 8 9 10 10 9 8 7 11 11 ...
##  $ tip.label  : chr [1:6] "t2" "t4" "t5" "t6" ...
##  $ edge.length: num [1:10] 0.951 0.45 0.996 0.525 0.198 ...
##  $ Nnode      : int 5
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "cladewise"
```

It has a "phylo" class, which it looks like is a fancy type of list with slots for edges, edge lengths, and labels. Often, these fancier types of objects will have useful methods for `print` and `summary`, so we can check these out:

```
tree1 %>%
  summary()
```

```
##
## Phylogenetic tree: .
##
##   Number of tips: 6
##   Number of nodes: 5
##   Branch lengths:
##     mean: 0.6196081
##     variance: 0.07445404
##     distribution summary:
##      Min.   1st Qu.    Median   3rd Qu.      Max.
## 0.1984559 0.4677910 0.6041142 0.8319966 0.9964703
##   No root edge.
##   Tip labels: t2
##               t4
##               t5
##               t6
##               t1
##               t3
##   No node labels.
```

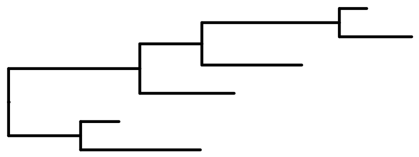
```
tree1 %>%
  print()
```

```
##
## Phylogenetic tree with 6 tips and 5 internal nodes.
##
## Tip labels:
## [1] "t2" "t4" "t5" "t6" "t1" "t3"
##
## Rooted; includes branch lengths.
```

In this case, `summary` doesn't help much, but `print` prints out all the values. It looks like we've got a pretty small tree in this case, with 5 nodes and 10 edges. It looks like that's all the data we've got stored in this object… I'm guessing that, normally, you'd process and analyze your data some to get to the point of what we have available here. Based on the text, these are data simulated by following the (known) rules for a certain tree.

To visualize this tree, you can use the `ggtree` function from the "ggtree" package. At minimum, it looks like this just shows the structure of the tree:

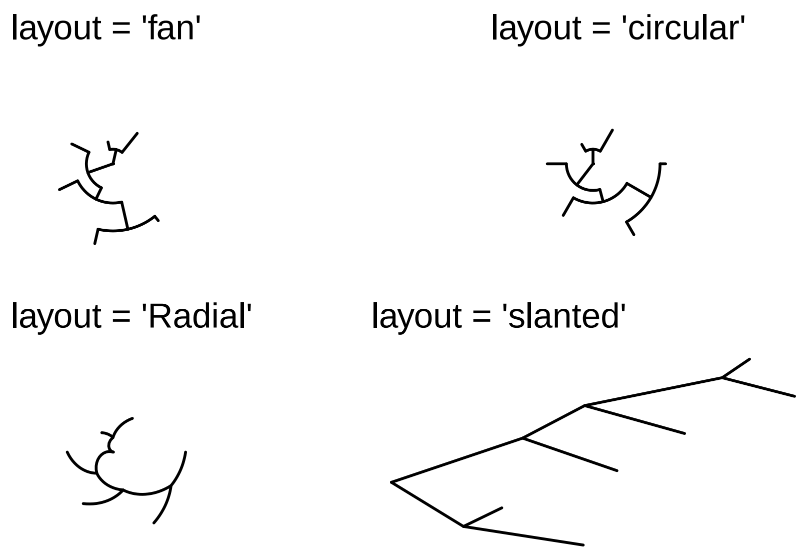
```
tree1 %>%
  ggtree()
```



Check out the helpfile for `ggtree` (`?ggtree`) to find out more about the options you can include to customize the output. These include:

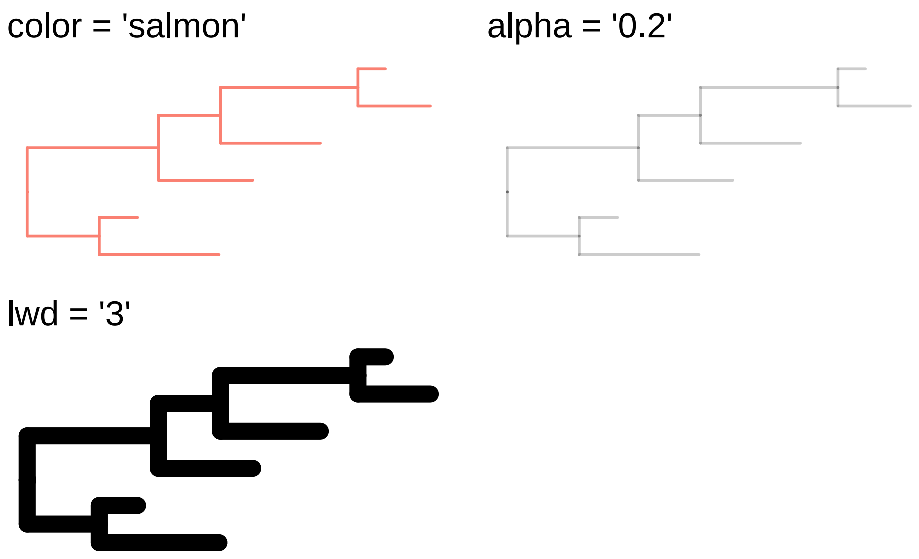
Printing the tree with a different structure:

```r
a <- tree1 %>%
  ggtree(layout = "fan") +
  ggtitle("layout = 'fan'")

b <- tree1 %>%
  ggtree(layout = "circular") +
  ggtitle("layout = 'circular'")

c <- tree1 %>%
  ggtree(layout = "radial") +
  ggtitle("layout = 'Radial'")

d <- tree1 %>%
  ggtree(layout = "slanted") +
  ggtitle("layout = 'slanted'")

library(gridExtra)
grid.arrange(a, b, c, d, nrow = 2)
```



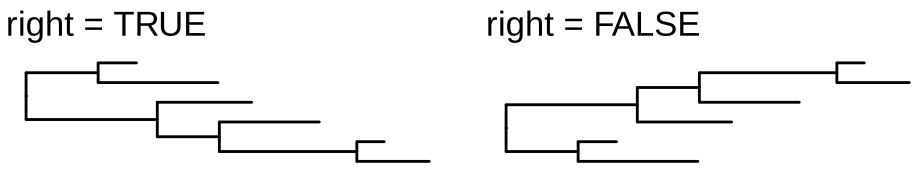
You can also include typical `ggplot` parameters, like `color`, `alpha`, and `lwd` (which stands for "line width"):

```r
# Change the color
a <- tree1 %>%
  ggtree(color = "salmon") +
  ggtitle("color = 'salmon'")

# Make somewhat transparent
b <- tree1 %>%
  ggtree(alpha = 0.2) +
  ggtitle("alpha = '0.2'")

# Make lines thicker
c <- tree1 %>%
  ggtree(lwd = 3) +
  ggtitle("lwd = '3'")

grid.arrange(a, b, c, nrow = 2)
```



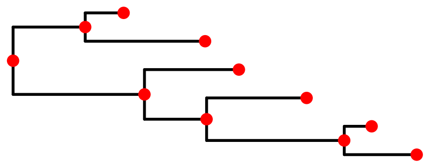
You can also specify where the smallest *clade* should fall with the `right` argument:

```r
a <- tree1 %>%
  ggtree(right = TRUE) +
  ggtitle("right = TRUE")

b <- tree1 %>%
  ggtree(right = FALSE) +
  ggtitle("right = FALSE")

grid.arrange(a, b, nrow = 1)
```



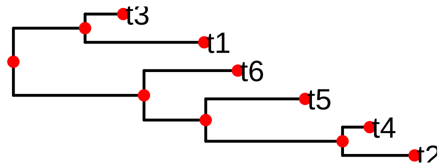
You can add on geoms to these trees using functions from `ggtree`, as well. For example, you can add on circles at the end of each branch using `geom_point`, using all the classic parameters you might want to use for `geom_point`:

```r
tree1 %>%
  ggtree(right = TRUE) +
  geom_point(color = "red")
```



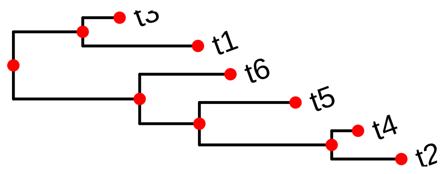
You can add labels at the ends of the branches with `geom_tiplab`:

```
tree1 %>%
  ggtree(right = TRUE) +
  geom_point(color = "red") +
  geom_tiplab()
```



This will use the values stored in the `tip.label` element of the original data object (`tree1`). This function (`geom_tiplab`) has a number of parameters that you can use to customize it, include the font size of the labels (`size`), their orientation (`angle`), and how far they are away from the tree (`offset`):

```
tree1 %>%
  ggtree(right = TRUE) +
  geom_point(color = "red") +
  geom_tiplab(angle = 20, offset = 0.1, size = 4)
```



Next, the text covers how to simulate data from a known tree. You can use the function `simSeq` from the “phangorn” package to do this.

By default, if you apply this to an object with the “phylo” class (like `tree1`), it will simulate sequences of length 1,000 from that tree. You can change the length of the simulated sequences with the argument `l` (e.g., `l = 60` will generate sequences of length 60). You can specify the frequencies of each base with the argument `bf`, where the order should be the frequencies of A, C, G, and then T.

```
library(phangorn)
sim_tree1 <- tree1 %>%
  simSeq(l = 60, # simulate sequences with 60 nucleotides
         bf = c(1 / 8, 1 / 8, 3 / 8, 3 / 8), # set the frequencies for A, C, G, and T, in order
         rate = 0.1 # set the mutation rate; must be larger than 0
         )
sim_tree1 %>%
  class()
```

```
## [1] "phyDat"
```

```
sim_tree1 %>%
  str()
```

```
## List of 6
##  $ t2: int [1:24] 2 3 2 2 2 4 3 3 3 1 ...
##  $ t4: int [1:24] 2 3 2 2 2 4 3 4 3 3 ...
##  $ t5: int [1:24] 2 3 2 4 2 4 3 4 3 3 ...
##  $ t6: int [1:24] 2 3 2 4 2 4 3 4 3 3 ...
##  $ t1: int [1:24] 3 3 3 2 2 4 3 4 4 3 ...
##  $ t3: int [1:24] 2 3 3 2 2 4 1 4 4 3 ...
##  - attr(*, "class")= chr "phyDat"
##  - attr(*, "weight")= int [1:24] 1 10 1 1 4 18 1 1 3 1 ...
##  - attr(*, "nr")= int 24
##  - attr(*, "nc")= num 4
##  - attr(*, "index")= int [1:60] 1 2 3 4 5 5 6 2 6 6 ...
##  - attr(*, "levels")= chr [1:4] "a" "c" "g" "t"
##  - attr(*, "allLevels")= chr [1:18] "a" "c" "g" "t" ...
##  - attr(*, "type")= chr "DNA"
##  - attr(*, "contrast")= num [1:18, 1:4] 1 0 0 0 0 1 1 1 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:4] "a" "c" "g" "t"
```

If you convert the results to a character class, you can see that you have a matrix filled with the characters “a”, “c”, “g”, and “t”, where each row proves one of the six simulated sequences and each column represents a position along that sequence:

```
sim_tree1 %>%
  as.character()
```

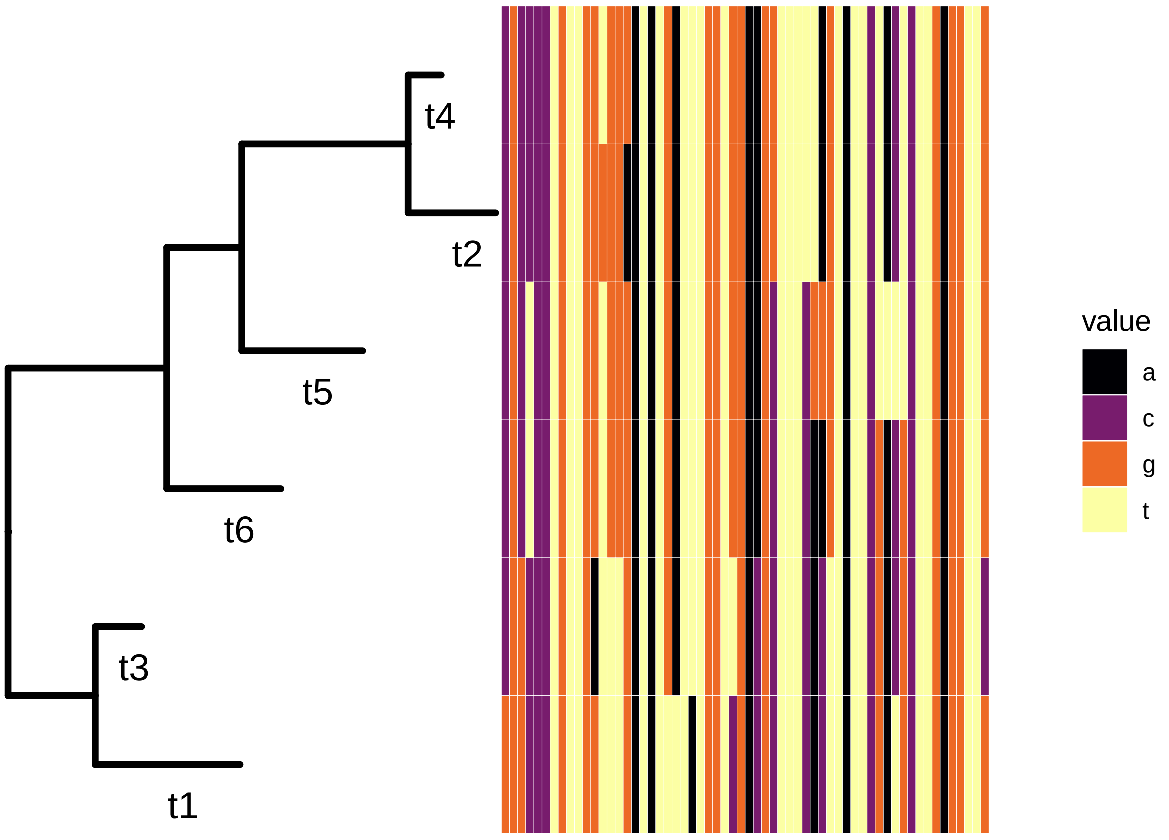
```
##    [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## t2 "c"  "g"  "c"  "c"  "c"  "c"  "t"  "g"  "t"  "t"   "g"   "g"   "g"   "g"  
## t4 "c"  "g"  "c"  "c"  "c"  "c"  "t"  "g"  "t"  "t"   "g"   "g"   "t"   "g"  
## t5 "c"  "g"  "c"  "t"  "c"  "c"  "t"  "g"  "t"  "t"   "g"   "g"   "t"   "g"  
## t6 "c"  "g"  "c"  "t"  "c"  "c"  "t"  "g"  "t"  "t"   "g"   "g"   "t"   "g"  
## t1 "g"  "g"  "g"  "c"  "c"  "c"  "t"  "g"  "t"  "t"   "g"   "g"   "t"   "t"  
## t3 "c"  "g"  "g"  "c"  "c"  "c"  "t"  "g"  "t"  "t"   "g"   "a"   "t"   "t"  
##    [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## t2 "g"   "a"   "a"   "t"   "a"   "t"   "g"   "a"   "t"   "t"   "t"   "g"  
## t4 "g"   "g"   "a"   "t"   "a"   "t"   "g"   "a"   "t"   "t"   "t"   "g"  
## t5 "g"   "g"   "a"   "t"   "a"   "t"   "g"   "a"   "t"   "t"   "t"   "g"  
## t6 "g"   "g"   "a"   "t"   "a"   "t"   "g"   "a"   "t"   "t"   "t"   "g"  
## t1 "t"   "g"   "a"   "t"   "a"   "t"   "t"   "t"   "t"   "a"   "t"   "g"  
## t3 "t"   "g"   "a"   "t"   "a"   "t"   "g"   "a"   "t"   "t"   "t"   "g"  
##    [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
## t2 "g"   "t"   "g"   "g"   "a"   "a"   "g"   "g"   "t"   "t"   "t"   "t"  
## t4 "g"   "t"   "g"   "g"   "a"   "a"   "g"   "g"   "t"   "t"   "t"   "t"  
## t5 "g"   "t"   "g"   "g"   "a"   "a"   "g"   "c"   "t"   "t"   "t"   "c"  
## t6 "g"   "t"   "g"   "g"   "a"   "a"   "g"   "c"   "t"   "t"   "t"   "c"  
## t1 "g"   "t"   "c"   "g"   "a"   "c"   "g"   "c"   "t"   "t"   "t"   "c"  
## t3 "g"   "t"   "t"   "g"   "a"   "c"   "g"   "c"   "t"   "t"   "t"   "c"  
##    [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
## t2 "t"   "a"   "g"   "t"   "a"   "t"   "t"   "c"   "t"   "a"   "c"   "t"  
## t4 "t"   "a"   "g"   "t"   "a"   "t"   "t"   "c"   "t"   "a"   "c"   "t"  
## t5 "g"   "g"   "g"   "t"   "a"   "t"   "t"   "c"   "t"   "t"   "t"   "t"  
## t6 "a"   "a"   "g"   "t"   "a"   "t"   "t"   "c"   "g"   "a"   "c"   "g"  
## t1 "a"   "c"   "t"   "t"   "a"   "t"   "t"   "c"   "g"   "a"   "t"   "g"  
## t3 "a"   "c"   "t"   "t"   "a"   "t"   "t"   "c"   "g"   "a"   "c"   "g"  
##    [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60]
## t2 "c"   "t"   "t"   "g"   "a"   "g"   "g"   "t"   "t"   "g"  
## t4 "c"   "t"   "t"   "g"   "a"   "g"   "g"   "t"   "t"   "g"  
## t5 "c"   "t"   "t"   "g"   "a"   "g"   "g"   "t"   "t"   "g"
## t6 "c"   "t"   "t"   "g"   "a"   "g"   "g"   "t"   "t"   "g"
## t1 "c"   "t"   "t"   "g"   "a"   "g"   "g"   "t"   "t"   "g"
## t3 "c"   "t"   "t"   "g"   "a"   "g"   "g"   "t"   "t"   "c"
```

To plot, you can conver this to a dataframe and then use `ggtree` with `gheatmap` (also from the "ggtree" package):

```r
sim_tree1 %>%
  as.character() %>%
  as.data.frame()
```

```
##    V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21
## t2  c  g  c  c  c  c  t  g  t   t   g   g   g   g   g   a   a   t   a   t   g
## t4  c  g  c  c  c  c  t  g  t   t   g   g   t   g   g   g   a   t   a   t   g
## t5  c  g  c  t  c  c  t  g  t   t   g   g   t   g   g   g   a   t   a   t   g
## t6  c  g  c  t  c  c  t  g  t   t   g   g   t   g   g   g   a   t   a   t   g
## t1  g  g  g  c  c  c  t  g  t   t   g   g   t   t   t   g   a   t   a   t   t
## t3  c  g  g  c  c  c  t  g  t   t   g   a   t   t   t   g   a   t   a   t   g
##    V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34 V35 V36 V37 V38 V39 V40
## t2   a   t   t   t   g   g   t   g   g   a   a   g   g   t   t   t   t   t   a
## t4   a   t   t   t   g   g   t   g   g   a   a   g   g   t   t   t   t   t   a
## t5   a   t   t   t   g   g   t   g   g   a   a   g   c   t   t   t   c   g   g
## t6   a   t   t   t   g   g   t   g   g   a   a   g   c   t   t   t   c   a   a
## t1   t   t   a   t   g   g   t   c   g   a   c   g   c   t   t   t   c   a   c
## t3   a   t   t   t   g   g   t   t   g   a   c   g   c   t   t   t   c   a   c
##    V41 V42 V43 V44 V45 V46 V47 V48 V49 V50 V51 V52 V53 V54 V55 V56 V57 V58 V59
## t2   g   t   a   t   t   c   t   a   c   t   c   t   t   g   a   g   g   t   t
## t4   g   t   a   t   t   c   t   a   c   t   c   t   t   g   a   g   g   t   t
## t5   g   t   a   t   t   c   t   t   t   t   c   t   t   g   a   g   g   t   t
## t6   g   t   a   t   t   c   g   a   c   g   c   t   t   g   a   g   g   t   t
## t1   t   t   a   t   t   c   g   a   t   g   c   t   t   g   a   g   g   t   t
## t3   t   t   a   t   t   c   g   a   c   g   c   t   t   g   a   g   g   t   t
##    V60
## t2   g
## t4   g
## t5   g
## t6   g
## t1   g
## t3   c
```

```r
# Create a ggplot object with the original tree
orig_tree <- tree1 %>%
  ggtree(lwd = 1.2) +
  geom_tiplab(aes(x = branch), size = 5, vjust = 2)

# Plot this, while adding on a heatmap showing the simulated sequences
library(viridis)
orig_tree %>%
  gheatmap(sim_tree1 %>%
             as.character() %>%
             as.data.frame() %>%
             select(1:60),
           offset = 0.01, colnames = FALSE) +
  scale_fill_viridis(option = "B", discrete = TRUE)
```

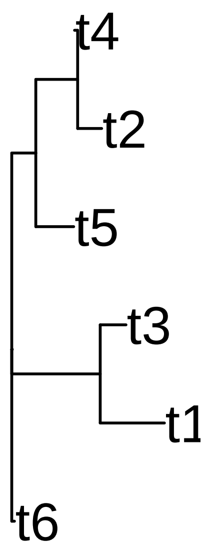


Next, they build a tree from these sequences that they just simulated from the original tree. They seem to use a maximum likelihood method for estimating the parameters of the tree, based on a model for molecular evolution called "JC68" that seems to be based on equal base frequencies (based on the help documentation). This method (`dist.ml`) determines the distances between each of the sequences, creating a distance matrix, which then is input into `nj` to create a tree base on a neighbor-joining algorithm:

```
tree_nj <- sim_tree1 %>%
  # From the sequences, get the pairwise distances, using
  # the "JC69" amino acid model, which uses equal base frequencies
  # (this is the default for `model`, so you could also exclude this in
  # the call)
  dist.ml(model = "JC69") %>%
  # Estimate a tree using a neighbor-joining algorithm
  nj()

tree_nj
```

```
##
## Phylogenetic tree with 6 tips and 4 internal nodes.
##
## Tip labels:
## [1] "t2" "t4" "t5" "t6" "t1" "t3"
##
## Unrooted; includes branch lengths.
```

```
tree_nj %>%
  ggtree() +
  geom_tiplab(size = 7) +
  xlim(0, 0.8)
```



Generate the maximum likelihood score of the original tree based on the simulated tree data:

```
# k gives the number of intervals of the discrete gamma distribution (?)
# The default is to assume a shape parameter of 1 for the gamma distribution
# and a rate of 1
pml(tree = tree1, data = sim_tree1, k = 4)
```

```
##
##  loglikelihood: -275.1399
##
## unconstrained loglikelihood: -152.3336
## Discrete gamma model
## Number of rate categories: 4
## Shape parameter: 1
##
## Rate matrix:
##   a c g t
## a 0 1 1 1
## c 1 0 1 1
## g 1 1 0 1
## t 1 1 1 0
##
## Base frequencies:
## 0.25 0.25 0.25 0.25
```

Next, try out with HIV GAG sequence data. This was a very big file, so I haven't moved it into this repo. Instead, I left it in my "Downloads" folder, where I'm reading it in here. I downloaded this data from https://www.hiv.lanl.gov/content/sequence/NEWALIGN/align.html, selecting "GAG" for the "region" and otherwise using all the default choices.

```
hiv_seq <- read.phyDat("~/Downloads/HIV1_ALL_2018_gag_DNA.fasta",
                       format = "fasta")
```

```
hiv_seq
```

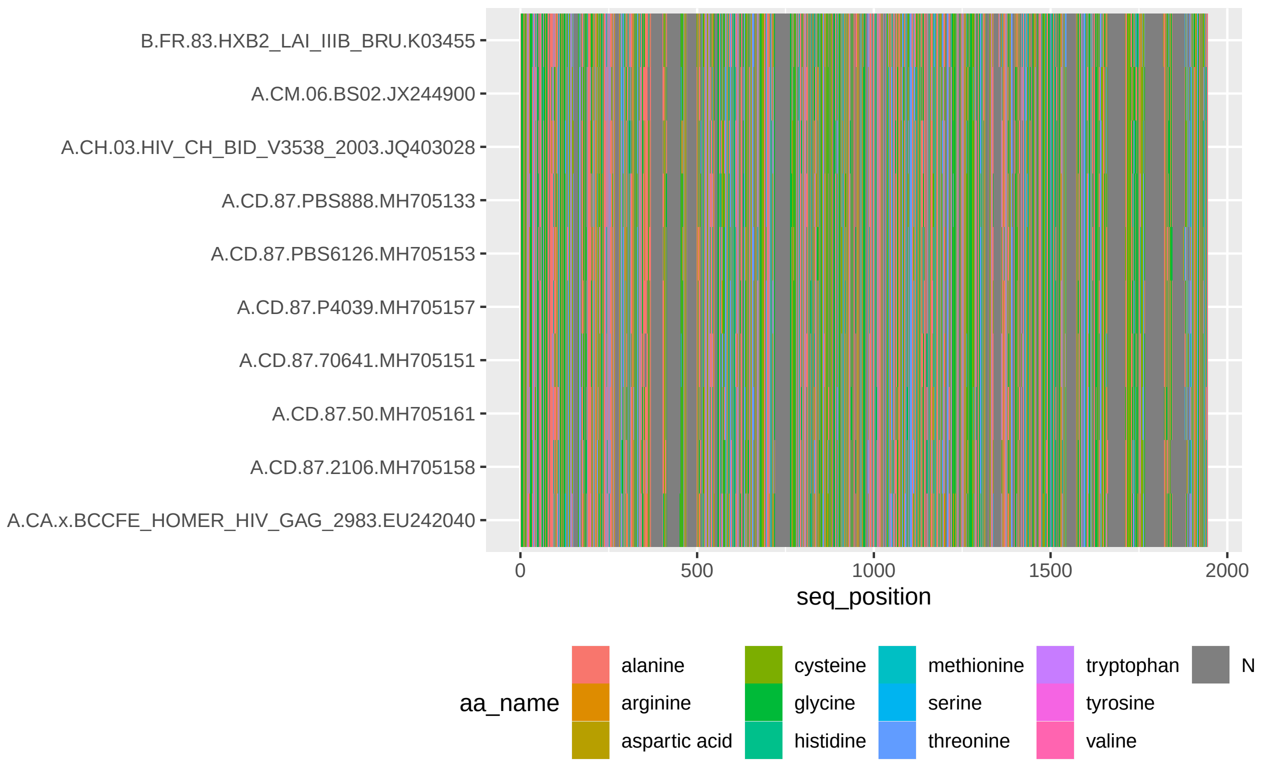
```
## 9547 sequences with 1944 character and 1877 different site patterns.
## The states are a c g t
```

```r
# Check out the structure of the first few sequences
hiv_seq %>%
  `[`(1:5) %>%
  str()
```

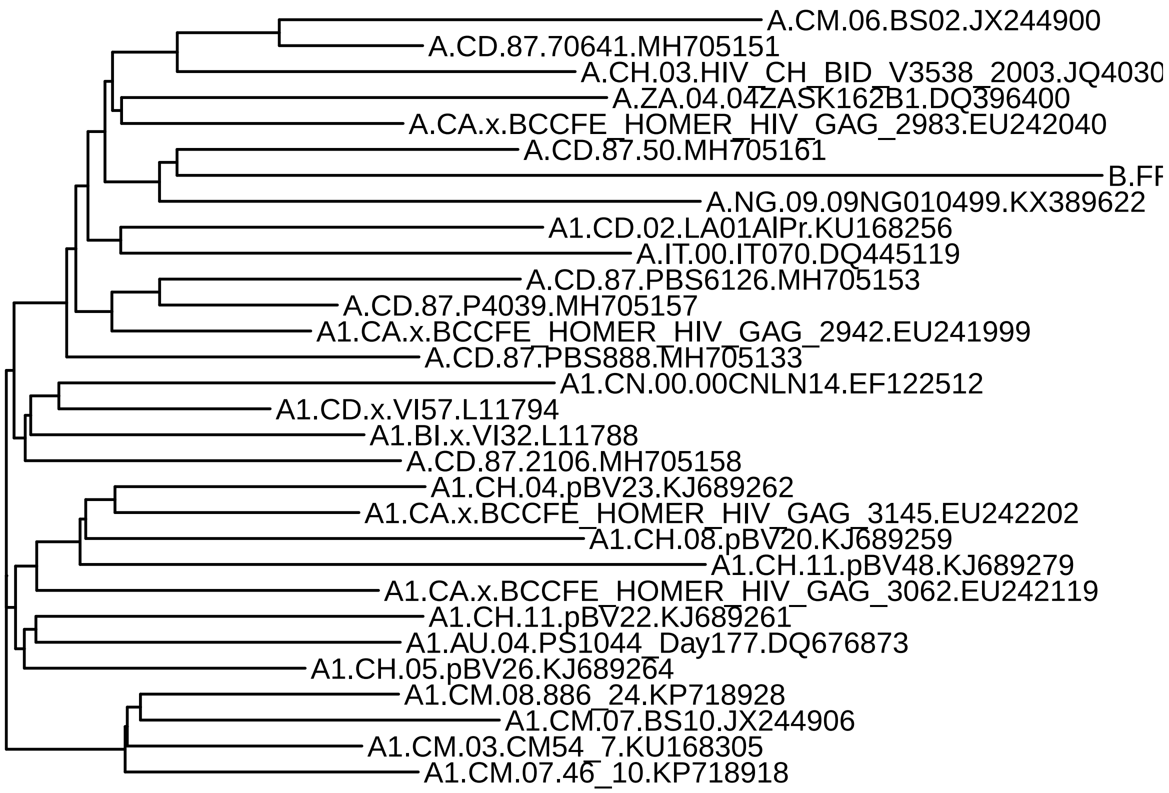
```
## List of 5
##  $ B.FR.83.HXB2_LAI_IIIB_BRU.K03455        : int [1:1877] 1 4 3 3 3 4 3 2 3 1 ...
##  $ A.CA.x.BCCFE_HOMER_HIV_GAG_2983.EU242040: int [1:1877] 1 4 3 3 3 4 3 2 3 1 ...
##  $ A.CD.87.2106.MH705158                   : int [1:1877] 1 4 3 3 3 4 3 2 3 1 ...
##  $ A.CD.87.50.MH705161                     : int [1:1877] 1 4 3 3 3 4 3 2 3 1 ...
##  $ A.CD.87.70641.MH705151                  : int [1:1877] 1 4 3 3 3 4 3 2 3 1 ...
##  - attr(*, "class")= chr "phyDat"
##  - attr(*, "weight")= int [1:1877] 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "nr")= int 1877
##  - attr(*, "nc")= num 4
##  - attr(*, "index")= int [1:1944] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "levels")= chr [1:4] "a" "c" "g" "t"
##  - attr(*, "allLevels")= chr [1:18] "a" "c" "g" "t" ...
##  - attr(*, "type")= chr "DNA"
##  - attr(*, "contrast")= num [1:18, 1:4] 1 0 0 0 0 1 1 1 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:4] "a" "c" "g" "t"
```

You could visualize this with `geom_tile` geoms (this is for the first 10 sequences):

```r
# Add in code for amino acid letters so we can make a better legend
aa_code <- tribble(
  ~ symbol, ~ aa_name,
  "c", "cysteine",
  "h", "histidine",
  "i", "isoleucine",
  "m", "methionine",
  "s", "serine",
  "v", "valine",
  "a", "alanine",
  "g", "glycine",
  "l", "leucine",
  "p", "proline",
  "t", "threonine",
  "r", "arginine",
  "f", "phenylalanine",
  "y", "tyrosine",
  "w", "tryptophan",
  "d", "aspartic acid",
  "n", "asparagine",
  "e", "glutamic acid",
  "q", "glutamine",
  "l", "lysine"
)

hiv_seq %>%
  `[`(1:10, ) %>%
  as_tibble(rownames = NULL) %>%
  mutate(seq_position = 1:n()) %>%
  pivot_longer(-seq_position, names_to = "sample_id", values_to = "amino_acid") %>%
  mutate(amino_acid = ifelse(amino_acid == "-", NA, amino_acid)) %>%
  left_join(aa_code, by = c("amino_acid" = "symbol")) %>%
  ggplot(aes(x = seq_position, y = sample_id, fill = aa_name)) +
  geom_tile() +
  theme(axis.title.y = element_blank(),
        legend.position = "bottom")
```



```
hiv_seq %>%
  # Limit to just a few of the sequences so it won't take forever
  `[`(1:30) %>%
  # Calculate the distance metric (using maximum likelihood)
  dist.ml() %>%
  # Make the tree from the distance matrix (using neighbor-joining)
  nj() %>%
  # Plot the tree
  ggtree() +
  # Add the labels at the end of the tree
  geom_tiplab()
```



To wrap up section 10.4.4., they give an application to 16S rRNA data. They have some data available in the book's datasets as an ".rds" file (a common binary R format), so you first need to read that data in. You can do that using `readRDS`:

```r
seqtab <- readRDS("data/seqtab.rds")
seqtab %>%
  dim()
```

```
## [1]  19 268
```

```r
seqtab %>%
  str()
```

```
##  int [1:19, 1:268] 554 401 432 265 210 394 611 301 1443 830 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:19] "F3D0" "F3D1" "F3D141" "F3D142" ...
##   ..$ : chr [1:268] "GCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGTGCGCAGGCGGAAGATCAAGTCAGCGGTAAAATTGAGAGGCTCAA
```

Next, we want to get a vector of sequences from the data we just read in. You can use the `getSequences` function from the `dada2` package to do that:

```r
# BiocManager::install("dada2")
library(dada2)

seqs <- seqtab %>%
  getSequences()

# This new object is a vector of character strings
seqs %>%
  class()
```

```
## [1] "character"
```

```r
seqs %>%
  length()
```

```
## [1] 268
```

```r
# You can see the first two strings (at least the start)
seqs[1:2]
```

```
## [1] "GCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGTGCGCAGGCGGAAGATCAAGTCAGCGGTAAAATTGAGAGGCTCAACCTCTTCGAGCCGTTGA
## [2] "GCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGTGCGCAGGCGGACTCTCAAGTCAGCGGTCAAATCGCGGGGCTCAACCCCGTTCCGCCGTTGA
```

Next, we evidently want to compare them to a dataset where sequences have been classified. This reference dataset for the book was evidently in a temporary file directory when they created the book (at least the text version; "/tmp"), and so that's not much help to us here. However, it looks like there's a version online here: https://zenodo.org/record/801828#.XqH4EdNKg6U You can download the file "rdp_train_set_16.fa.gz" from that page and proceed from there.

This is a fasta reference file, so the function to read it in is different than most files we've worked with.

```r
# This might take a few seconds to run...
# If you're struggling with this, it looks like they've pre-run to this step,
# and then saved the output as "taxtab16.rds" in the book's data folder
taxtab <- seqtab %>%
  assignTaxonomy(refFasta = "~/Downloads/rdp_train_set_16.fa.gz")

taxtab %>%
  dim()
```

```
## [1] 268   6
```

```r
# It's a matrix
taxtab %>%
  class()
```

```
## [1] "matrix"
```

```r
# The rownames seem to be sequences
taxtab %>%
  rownames() %>%
  `[`(1:3)
```

```
## [1] "GCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGTGCGCAGGCGGAAGATCAAGTCAGCGGTAAAATTGAGAGGCTCAACCTCTTCGAGCCGTTGA
## [2] "GCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGTGCGCAGGCGGACTCTCAAGTCAGCGGTCAAATCGCGGGGCTCAACCCCGTTCCGCCGTTGA
## [3] "GCGAGCGTTATCCGGATTTATTGGGTTTAAAGGGTGCGTAGGCGGGCTGTTAAGTCAGCGGTCAAATGTCGGGGCTCAACCCCGGCCTGCCGTTGA
```

```r
# If you convert to a tibble, by default it will take off the rownames. Then
# you can use `slice` to show just a few rows
taxtab %>%
  as_tibble() %>%
  slice(1:6)
```

```
## # A tibble: 6 x 6
##   Kingdom  Phylum        Class       Order         Family             Genus
##   <chr>    <chr>         <chr>       <chr>         <chr>              <chr>
## 1 Bacteria Bacteroidetes Bacteroidia Bacteroidales Porphyromonadaceae <NA>
## 2 Bacteria Bacteroidetes Bacteroidia Bacteroidales Porphyromonadaceae <NA>
## 3 Bacteria Bacteroidetes Bacteroidia Bacteroidales Porphyromonadaceae <NA>
## 4 Bacteria Bacteroidetes Bacteroidia Bacteroidales Porphyromonadaceae Barnesiel~
## 5 Bacteria Bacteroidetes Bacteroidia Bacteroidales Bacteroidaceae     Bacteroid~
## 6 Bacteria Bacteroidetes Bacteroidia Bacteroidales Porphyromonadaceae <NA>
```

Perform multiple alignment on the `seqs` data:

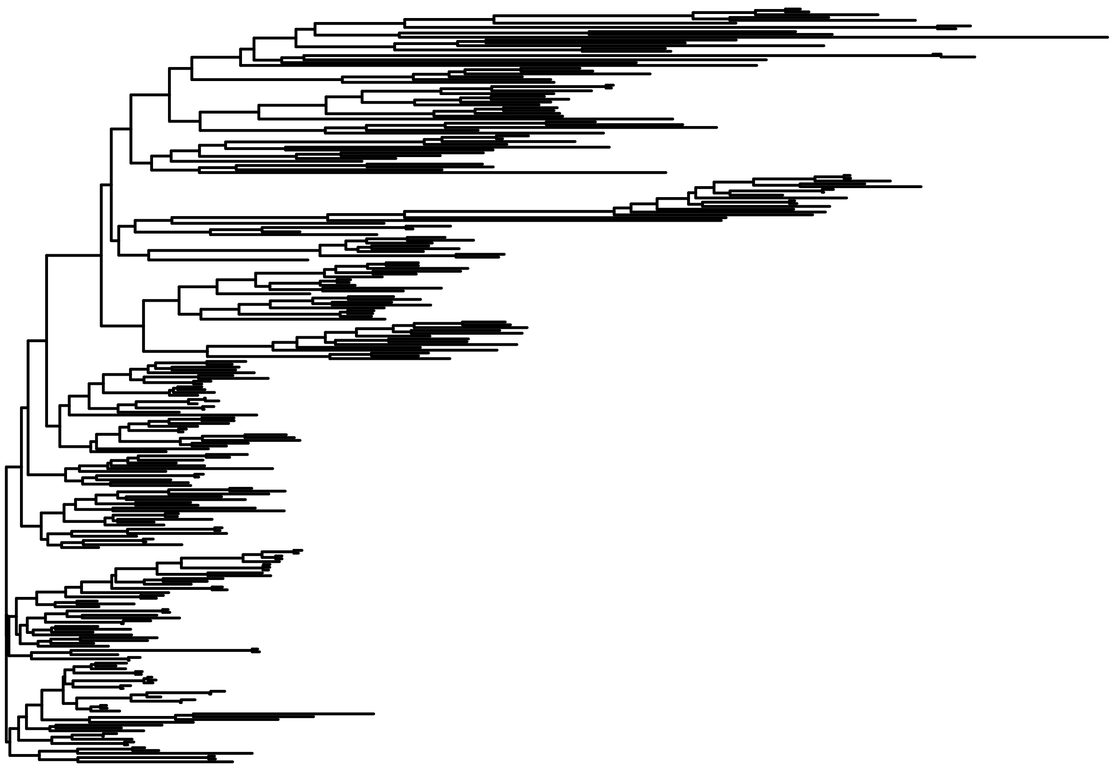
```r
# BiocManager::install("DECIPHER")
library(DECIPHER)

# Give each element in `seqs` the same sequence as the name, so we'll
# retain that through the analysis
names(seqs) <- seqs

alignment <- seqs %>%
  # Put the data (current in a vector of character strings) into a DNAStringSet
  # class
  DNAStringSet() %>%
  # Align all the sequences using a guide tree. We're specifying not to use an
  # anchor (`anchor = NA`) and not to print out updates as it goes (`verbose = FALSE`)
  AlignSeqs(anchor = NA, verbose = FALSE)

# The result is a DNAStringSet, but now it's been aligned across all the sequences
alignment %>%
  class()
```

```
## [1] "DNAStringSet"
## attr(,"package")
## [1] "Biostrings"
```

```r
treeNJ <- alignment %>%
  as.matrix() %>%
  # Change into a `phyDat` object (must be a matrix first)
  phyDat(type = "DNA") %>%
  # Calculate the distance matrix using maximum likelihood
  dist.ml() %>%
  # Determine the tree from the distance matrix using neighbor-joining
  NJ()
```

You can see that this is a pretty massive tree:

```r
treeNJ %>%
  ggtree()
```



```r
fit <- treeNJ %>%
  pml(data = alignment %>% as.matrix() %>% phyDat(type = "DNA"))
fit
```

```
##
##  loglikelihood: -16305.83
##
## unconstrained loglikelihood: -1120.729
##
## Rate matrix:
##   a c g t
## a 0 1 1 1
## c 1 0 1 1
## g 1 1 0 1
## t 1 1 1 0
##
## Base frequencies:
## 0.25 0.25 0.25 0.25
```

```r
fitGTR <- fit %>%
  update(k = 4, inv = 0.2) %>%
  optim.pml(model = "GTR", optInv = TRUE, optGamma = TRUE,
            rearrangement = "stochastic", control = pml.control(trace = 0))
```

Some from section 10.2

It's helpful to get an idea of how you can express a graph in a data object in R. one was is with a matrix that gives all the connections between nodes (the edges):

```r
# Make a matrix of the connections (edges) between nodes, labeled as
# 1, 2, 3, 4, 5, 6
edges1 <- c(1, 3,
            2, 3,
            3, 4,
            4, 6,
            4, 5) %>%
  matrix(byrow = TRUE, ncol = 2)
edges1
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    3
## [3,]    3    4
## [4,]    4    6
## [5,]    4    5
```

Now, you can convert this to an object of the "igraph" class with:

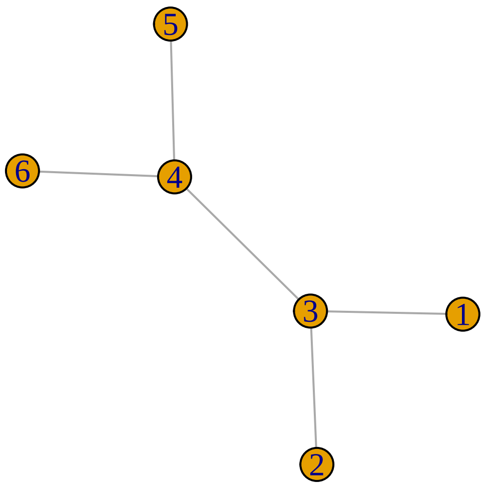
```r
# install.packages("igraph")
library(igraph)

g1 <- edges1 %>%
  graph_from_edgelist(directed = FALSE)

g1
```

```
## IGRAPH 4b02fe4 U--- 6 5 --
## + edges from 4b02fe4:
## [1] 1--3 2--3 3--4 4--6 4--5
```

```r
g1 %>%
  class()
```

```
## [1] "igraph"
```

```r
# There's a plot method for this class of object
g1 %>%
  plot()
```



Another object type that can store graph data is the `network` class. There's an example of data in this form in the `networksis` package called "finch":

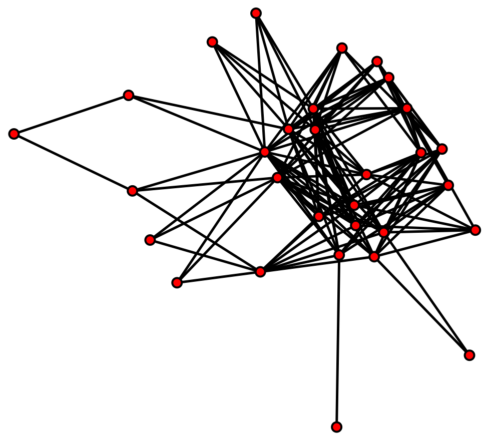
```r
# install.packages("networksis")
library(networksis)
data(finch)

finch %>%
  class()
```

```
## [1] "network"
```

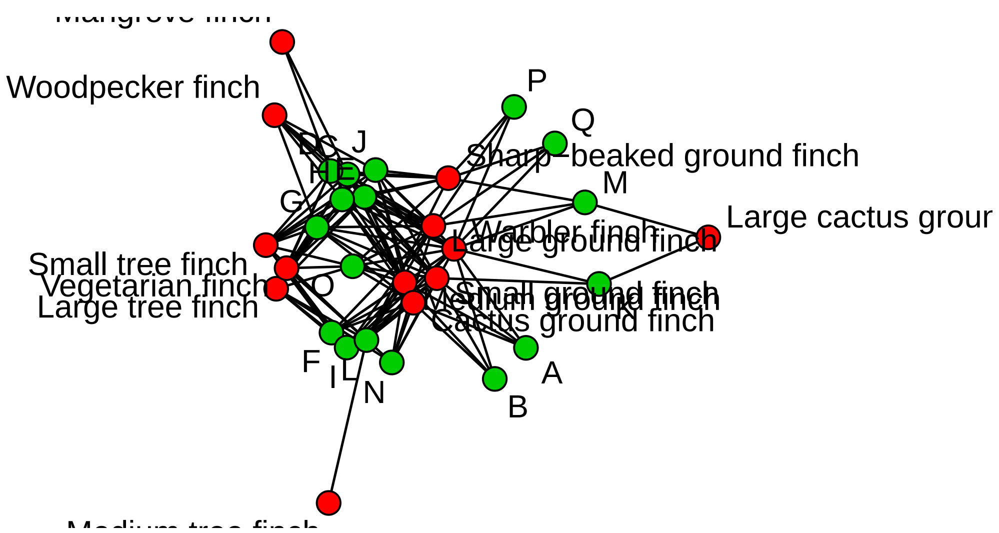
```r
finch
```

```
##  Network attributes:
##   vertices = 30
##   directed = FALSE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = 13
##   total edges= 122
##     missing edges= 0
##     non-missing edges= 122
##
##  Vertex attribute names:
##     vertex.names
##
## No edge attributes
```

```r
finch %>%
  plot()
```



```r
finch %>%
  plot(vertex.col = c(rep(2, 13), rep(3, 17)),
       vertex.cex = 2.5,
       displaylabels = TRUE)
```



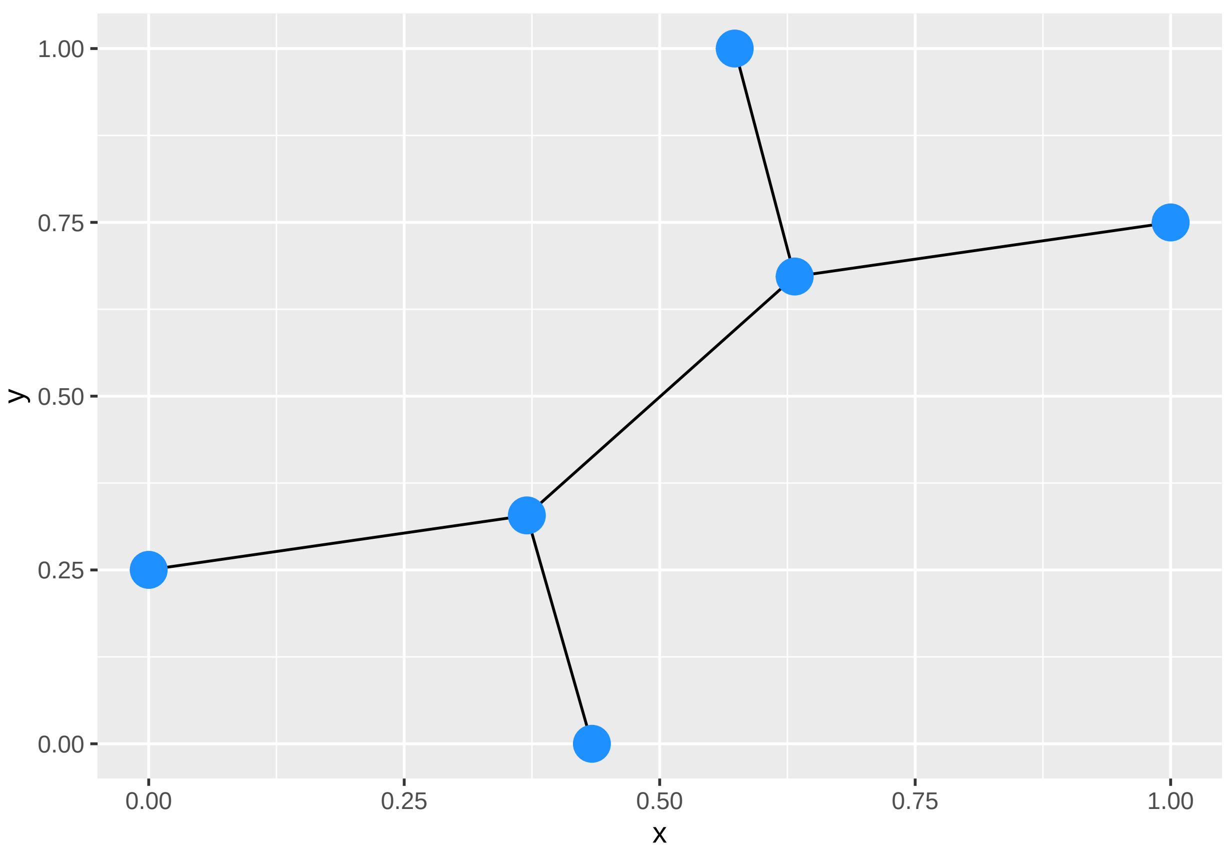
There are a number of different packages in R for visualizing graphs and networks. Another is `ggnetwork`, which does this in a ggplot framework. It has a function called `ggnetwork` that will convert data from an `igraph` object into a tidy dataframe, with values for how to locate the nodes and edges in the plot:

```
# install.packages("ggnetwork")
library(ggnetwork)

g1 %>%
  ggnetwork() %>%
  head()
```

```
##           x         y      xend      yend
## 3 0.3404995 0.6461010 0.0000000 0.6337874
## 4 0.3404995 0.6461010 0.3061406 1.0000000
## 5 0.6621337 0.3542557 0.3404995 0.6461010
## 6 0.7031002 0.0000000 0.6621337 0.3542557
## 7 1.0000000 0.3761802 0.6621337 0.3542557
## 1 1.0000000 0.3761802 1.0000000 0.3761802
```

You can then plot this object using `ggplot`. You'll use `geom_edges` to add the edges and customize their appearance and then `geom_nodes` to add the nodes and customize their appearance:

```
g1 %>%
  ggnetwork() %>%
  rownames_to_column(var = "vertex.names") %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_edges() +
  geom_nodes(aes(x = x, y = y), size = 6, color = "dodgerblue")
```


1.00
0.75
0.50
0.25
0.00
y
0.00
0.25
0.50
0.75
1.00
x