SPIDER: Species Identity and Evolution in R. A tutorial

Samuel D. J. Brown
s_d_j_brown@hotmail.com

September 8, 2011

1 Introduction

SPIDER : Species Identity and Evolution in R is an R package that implements a number of the most useful analyses for DNA barcoding studies and other research on species delimitation and speciation. It has been developed primarily by staff and students in the Department of Ecology at Lincoln University, and was first released in 2011. The package seeks to a number of analyses that the developers have found useful in their own research and hadn't been implemented in other R packages.

This document is a tutorial on the use of some of the functions implemented in SPIDER and guides to the interpretation of its output. It is intended to be useful for the absolute beginner, and users who are familiar with R and APE are encouraged to skip the first few sections. This tutorial does not discuss in detail the basic R syntax and data structures (with a few exceptions), and the reader is referred to other manuals[1] for that information.

1.1 Conventions

1.1.1 Species

The term 'species' will be used a lot in this document. The creators of the package are biologists, mostly working with DNA sequences in a taxonomic and systematic context, and so this term is naturally the one we feel most comfortable using. However, it is important to remember that 'species' in this context is not necessarily limited to biological species, but can refer to any group, population or other form of cluster being dealt with.

[1] The freely available An Introduction to R, Using R for data analysis and graphics and R for beginners are particularly recommended

1.1.2 R commands

Commands to be typed into the R console are typeset in `blue typewriter` font, while output are typeset in `red typewriter` font. Functions mentioned in the text are typeset in `black typewriter` font.

1.1.3 Vectors

Character vectors will be discussed a lot, particularly in "Species vectors" (Section 4), so I will briefly discuss them here. Vectors form the basic R data structure, created in the following way:

```
(x <- c(1, 4, 6, 7, 2))
[1] 1 4 6 7 2
(y <- c("A", "B", "C", "F", "G"))
[1] "A" "B" "C" "F" "G"
```

There are many different modes of vectors, however here we will only deal with two of them: numeric and character. The first example above is a numeric vector—funnily enough, it's made up of numbers. The second is a character vector. Numbers and letters can be mixed, but the result will be a character vector. The parts of a vector will be called 'elements' here, such that the first element of `x` is 1, while that of `y` is "A". The 2nd is 4 and "B" respectively, and the 4th is 7 and "F".

1.1.4 Lists

SPIDER uses lists a lot. Lists are very useful in that they are able to store in one big object lots of other little objects bits of different classes, modes and lengths. In addition, the `lapply()` function makes it easy to do the same thing to each element within a list. To demonstrate the utility of lists:

```
testList <- list(a=1:5, b=LETTERS[1:10], c=matrix(1:4, 2))
testList


$a
[1] 1 2 3 4 5

$b
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"

$c
     [,1] [,2]
```

```
[1,]    1    3
[2,]    2    4
```

`testList` contains three quite different objects, all happily coexisting. These objects can be indexed in number of ways:

```
testList$a
[1] 1 2 3 4 5
testList[[1]]
[1] 1 2 3 4 5
testList[[2]][3:5]
[1] "C" "D" "E"
testList$c[2,]
[1] 2 4
```

1.1.5 Getting help

R is blessed in having a large community of users that willingly offer help free of charge through avenues such as mailing lists and Stack Overflow. HOWEVER! Before even thinking of asking a question there, try looking up the documentation that comes with the program. Simply put a question mark before the function name you have a problem with and a wealth of information will appear.

```
?read.dna
?sppVector
```

R help pages have a justified reputation for being dense and overwhelming at first glance, but keep trying because most of the time (particularly for the base packages) the answer is in there. It's jolly difficult writing help pages, and my attempts at it renew my appreciation for what others have produced.

If any of SPIDER's help pages fail to live up to standards, please send us an email, preferably with suggestions, and we'll update it accordingly.

2 Obtaining spider

SPIDER is a package of the statistical programming environment R, which is available for all computing platforms from the Comprehensive R Archive Network (CRAN, http://cran.r-project.org). A stable version of SPIDER is also available on CRAN, and can be downloaded from within R while connected to the internet by entering the following command at the prompt:

```
install.packages("spider")
```

In addition to the stable version on CRAN, a development version is available at R-Forge (http://spider.r-forge.r-project.org/). This version can be installed from within R by using the command:

```
install.packages("spider", repos="http://R-Forge.R-project.org")
```

SPIDER requires the installation of the packages APE (Paradis et al., 2004) and PEGAS (Paradis, 2010) which provide the primary data structures for working with DNA sequences and phylogenetic trees. If these packages are not already on your system, they will automatically be installed when the commands above have been run.

3 Loading data

3.1 Loading spider

Once you have installed SPIDER, load the package using the following command:

```
library(spider)
```

As well as SPIDER, this command will load the packages APE, PEGAS and the required packages for PEGAS, ADEGENET (Jombart, 2008) and MASS (Venables & Ripley, 2002).

3.2 Bundled datasets

Once the packages are loaded, you are ready to go! It's now time to get some data to play with. Included in SPIDER are two datasets including sequences from (appropriately enough) two New Zealand spider genera, the wolf spider genus *Anoteropsis* and the nurseryweb spider genus *Dolomedes*. These datasets have been published as Vink & Paterson (2003) and Vink & Dupérré (2010). To load up these datasets, simply type:

```
data(anoteropsis)
data(dolomedes)
```

More information about these datasets can be discovered by having a look at these objects:

```
anoteropsis
```

```
33 DNA sequences in binary format stored in a list.

All sequences of same length: 409

Labels: Artoria_flavimanus Artoria_separata Anoteropsis_adumbrata_CO
Anoteropsis_adumbrata_BP Anoteropsis_aerescens_MK Anoteropsis_aerescens_TK ...

Base composition:
    a     c     g     t
0.261 0.135 0.159 0.445


class(anoteropsis)
[1] "DNAbin"
dolomedes

37 DNA sequences in binary format stored in a list.

All sequences of same length: 850

Labels: minorD003 minorD031 minorD026 minorD030D056 minorD006
minorD014D017D042D043 ...

Base composition:
    a     c     g     t
0.254 0.123 0.181 0.443


class(dolomedes)
[1] "DNAbin"
```

This tells us a whole bunch of stuff. Starting from the top, the first line tells us how many sequences are in the alignment, and if these sequences are stored as a list or as a matrix. This latter piece of information is handy to know if we want to manipulate them further down the track. The next line tells us the length of the sequences, including the minimum, maximum and average if the sequences are of different lengths. The third line shows the names of the first few sequences to give us an idea of the naming scheme, and finally the base composition of the alignment is given.

3.3 Loading your own

While the above datasets give us something to work on for the rest of the tutorial, what is a lot more interesting is working on your own datasets. To load your own datasets into SPIDER, use the function `read.dna()` supplied by APE.

This function can read DNA sequence files encoded in the PHYLIP sequential or interleaved formats, CLUSTAL, or (my preferred option) FASTA format. To load sequences stored in a FASTA formatted file called 'mySequences.fas' in the 'R' folder in your 'My Documents', use the following commands:

```
setwd("C:/My Document/R/")
dat <- read.dna("mySequences.fas", format="fasta")
dat
```

The first line (`setwd()`) sets the working directory for the R session. Every other file that you read or write for the duration of your session will end up in the directory specified here (unless you change it later in the session).

3.4 Getting sequences from GenBank

GenBank is the premier global depository for DNA sequences from all organisms. It is a huge and exciting place to visit and is an extremely valuable resource for doing all sorts of research that involves DNA.

APE provides a function `read.GenBank` which allows sequences to be downloaded from the system directly into a `DNAbin` object. The original version did not include information such as species names or gene regions, so a modified version is included in SPIDER, `read.GB`. This function works on a character vector of GenBank accession numbers, and retrieves the records that correspond to those numbers. As an example, the following code downloads 63 sequences of antarctic springtails (Greenslade et al., 2011).

```
seq <- 732028:732089
seq <- paste("HQ", seq, sep="")
seq <- c(seq, "DQ309568")
collembola <- read.GB(seq)
```

GenBank considers the DNA sequences it stores as individuals, so the data downloaded from it are not necessarily aligned. To check properly, it is necessary to export the sequences to an external alignment editor such as MEGA Tamura et al. (2011), 4Peaks or Seaview (Guoy et al., 2010). However, SPIDER provides a few tools for us to do a rough-and-ready job within R itself. First of all, we'll see how long the sequences are:

```
collLength <- sapply(collembola, length)
table(collLength)
```

```
560 614 618 619 624 630 631 640 658
  1   1   1   2   1   1  10   1  45
```

The majority of sequences are 658 bp in length. We can then retain all the sequences of this length in the alignment:

```
collembola <- collembola[which(collLength == 658)]
```

The function `seeBarcode()` produces a plot that represents each base as a coloured vertical line corresponding to its nucleotide. We can use this to give us an idea if these sequences are correctly aligned or not.

```
layout(matrix(1:6, ncol=1))
par(mar=c(0.5, 0, 0.5, 0))
apply(as.matrix(collembola)[sample(1:45,5),], MARGIN=1, FUN=seeBarcode)
seeBarcode(as.matrix(dolomedes)[sample(1:37, 1),1:658])
```

This code plots the sequences for 5 randomly sampled `collembola` and one `dolomedes`. The results can be seen in Figure 1. The top 5 all show reasonably similar patterns of variation. In particular, the blue, yellow & red, blue pattern in the second quarter is consistent across all `collembola` sequences. Compared to the `dolomedes` sequence, the difference is clear. For the purpose of the rest of the document, I'll assume the sequences in `collembola` are correctly aligned. As mentioned above however, in other analyses it will be necessary to determine this with greater certainty.

3.5 Getting sequences from BOLD

The Barcode of Life Data System (BOLD) is the data storage portal for a global endeavour to catalogue the living world, using short (around 500 bp) DNRA sequences from standard gene regions (CITE). The bulk of this data comes from the mitochondrial gene region cytochrome *c* oxidase I (COI). There are a lot of sequences in the database, but unfortunately only a small portion of them are currently made public. SPIDER contains functions for searching and downloading sequences from BOLD. The sequences being downloaded are from *Trigonopterus* weevils from northern New Guinea rainforest which show very high levels of diversity (Riedel et al., 2010; Riedel, 2010)

```
nums <- search.BOLD("Trigonopterus")
weevils <- read.BOLD(nums)
```

The function `search.BOLD()` searches the database and retrieves the specimen numbers for the records matching the search term. These numbers are then used by `read.BOLD()` to download the sequences.

NB: At the moment, the eFetch system for downloading sequences (used by `read.BOLD`) returns the same records three times. I emailed the good people at BOLD about it, but it is yet to be rectified. The line below will remove these triplicate records, but hopefully the problem will be fixed in the near future.

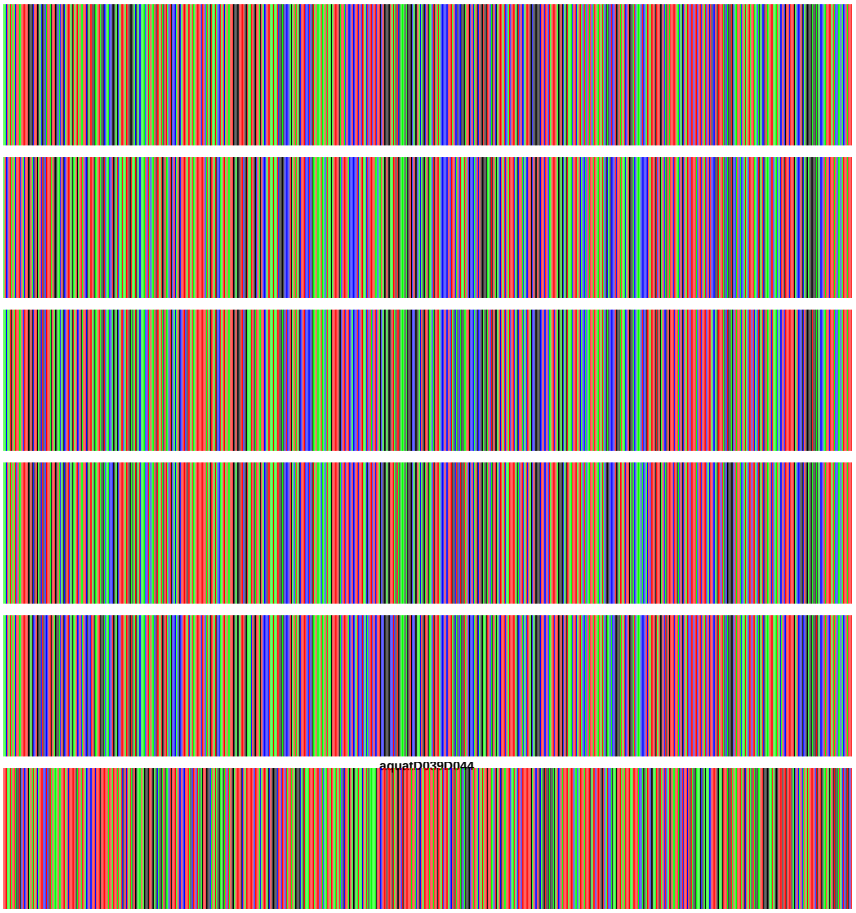

Figure 1: Five random `collembola` sequences and one from `dolomedes` to determine if sequences are roughly in alignment

```
weevils <- weevils[match(unique(names(weevils)), names(weevils))]
```

If you have a look at the `weevils` object, you'll see that the sequence length is given as 1500 bp. Later on though, we'll find that most of this length is made up of missing data, so we'll cut it down to size.

```
weevils <- as.matrix(weevils)[,1:700]
```

We now have an alignment that is only 670 bp long. Less than half of the original size. Much better!

4 Species vectors

4.1 What they're all about

Species vectors are the method used by SPIDER to distinguish between the groups in the dataset. They are simply a character vector that is the same length as the number of individuals in the dataset. The elements in the species vector correspond to the individuals in the dataset, so order is very important. The elements must also be *consistent within* each species and *unique between* them. In the terminology for this document, 'species vector' refers to the whole object, while 'species index' refers to individual elements within it.

Examples of species vectors and different methods of creating them are discussed below.

4.2 A basic species vector

```
set.seed(13)
tr <- rtree(9)
speciesVector <- c("A", "A", "B", "B", "C", "C", "D", "D", "D")
plot(tr)
tiplabels(speciesVector, adj=c(-2,0.5))
```

4.3 Extracting the species vector from pre-named labels

Most of the biologists I've come across name their sequences with a system that incorporates the species identity into it in some way, shape or form. The `anoteropsis` and `dolomedes` datasets demonstrate two different methods of incorporating the information into the labels.

4.3.1 `anoteropsis` species vector

```
head(names(anoteropsis))
```

```
[1] "Artoria_flavimanus"       "Artoria_separata"
[3] "Anoteropsis_adumbrata_CO" "Anoteropsis_adumbrata_BP"
[5] "Anoteropsis_aerescens_MK" "Anoteropsis_aerescens_TK"
```

In `anoteropsis` the labels are made up of the genus and species separated by an underscore. When more than one specimen per species is included, the geographic locality is also included, separated again by an underscore. The species vector can easily be constructed by splitting apart each element at the underscore (using `strsplit()`), and pasting the first two parts together again (using a combination of `paste()` and `sapply()`). To put it all together:

```
aa <- strsplit(names(anoteropsis), split="_")
anoSpp <- sapply(aa, function(x) paste(x[1], x[2], sep="_"))
head(anoSpp)
```

```
[1] "Artoria_flavimanus"    "Artoria_separata"      "Anoteropsis_adumbrata"
[4] "Anoteropsis_adumbrata" "Anoteropsis_aerescens" "Anoteropsis_aerescens"
```

4.3.2 `dolomedes` species vector

```
names(dolomedes)
```

```
[28] "aquatD054"             "dondaD012D013D035"     "dondaD034"
[31] "dondaD011"             "dondaD032D050"         "dondaD037"
[34] "dondaD052"             "dondaD053"             "schauD007"
```

The labels given to `dolomedes` are slightly different. The species names are abbreviated at the beginning of the label, followed by the specimen numbers possessing that haplotype. We can see though, that each species name is a set length—5 characters long. This allows us to use `substr()` to extract this information:

```
doloSpp <- substr(names(dolomedes), 1, 5)
doloSpp
```

```
 [1] "minor" "minor" "minor" "minor" "minor" "minor" "minor" "minor" "minor"
[10] "minor" "minor" "minor" "minor" "minor" "minor" "minor" "aquat" "aquat"
[19] "aquat" "aquat" "aquat" "aquat" "aquat" "aquat" "aquat" "aquat" "aquat"
[28] "aquat" "donda" "donda" "donda" "donda" "donda" "donda" "donda" "schau"
[37] "schau"
```

4.3.3 `collembola` species vector

```
names(collembola)
```

```
 [1] "HQ732028 | Anurida_maritima"
 [2] "HQ732029 | Ceratophysella_cf._gibbosa_CADH-2011"
 [3] "HQ732030 | Ceratophysella_denticulata"
 [4] "HQ732031 | Ceratophysella_denticulata"
 [5] "HQ732032 | Ceratophysella_denticulata"
```

The species vector for the `collembola` sequences we downloaded from GenBank can be retrieved in two ways. First, the harder way, which involves splitting the `names()` of the alignment at the pipe:

```
collSpp <- strsplit(names(collembola), split=" \\| ")
collSpp <- sapply(collSpp, function(x) x[2])
```

```
 [1] "Anurida_maritima"
 [2] "Ceratophysella_cf._gibbosa_CADH-2011"
 [3] "Ceratophysella_denticulata"
 [4] "Ceratophysella_denticulata"
 [5] "Ceratophysella_denticulata"
```

Note the double slash in front of the "`|`". This is because the "`|`" is a restricted character in R and the double slash tells it that we actually mean the character, not the symbol.

The easy way is to make use of the `attr()` slot that `read.GB()` tacks on the end of the object:

```
attr(collembola, "species")
```

Which does exactly the same thing.

4.3.4 `weevils` species vector

```
names(weevils)
```

```
 [1] "GBCL5258-09|Trigonopterus_sp._spz"  "GBCL5259-09|Trigonopterus_sp._sph"
 [3] "GBCL5260-09|Trigonopterus_sp._sph"  "GBCL5261-09|Trigonopterus_sp._spw"
 [5] "GBCL5262-09|Trigonopterus_sp._spw"  "GBCL5263-09|Trigonopterus_sp._spv"
 [7] "GBCL5264-09|Trigonopterus_sp._spv"  "GBCL5265-09|Trigonopterus_sp._spr"
```

To get the species vector from `weevils`, we could just grab the last part of the name after splitting on the underscore:

```
weevilSpp1 <- strsplit(names(weevils), split="_")
weevilSpp1 <- sapply(weevilSpp1, function(x) x[3])
```

```
 [1] "spz"  "sph"  "sph"  "spw"  "spw"  "spv"  "spv"  "spr"  "spr"  "spz"
[11] "spb"  "spae" "spae" "spw"  "spw"  "spv"  "spv"  "spav" "spav" "spc"
```

However, to make things a bit more informative I want to include the "Trigonopterus" part as well, though I want to drop the "sp." part. This will involve splitting on both the pipe and the underscore:

```
weevilSpp2 <- strsplit(names(weevils), split="[\\|_]")
weevilSpp2 <- sapply(weevilSpp2, function(x) paste(x[2], x[4], sep="-"))
```

```
[1] "Trigonopterus-spz" "Trigonopterus-sph" "Trigonopterus-sph"
[4] "Trigonopterus-spw" "Trigonopterus-spw" "Trigonopterus-spv"
```

The function `read.BOLD()` has an `attr()` slot in the same way that `read.GB()` has. However, with BOLD's current penchant for returning three of the same thing, the command `attr(weevils, "species"` returns a vector three times longer than the actual number of sequences. As species vectors need to be the same length as the number of individuals in the analysis, this is not a good thing.

4.4 Other useful functions for extracting names

The above examples have made extensive use of a few of the R's key functions for manipulating character strings. Other functions worth knowing about include the `grep()` family of functions, including `gsub()`. These functions have the ability to search for patterns in strings, and the latter has the ability to make changes to the strings along the way.

5 Sliding windows

5.1 The basic idea

The principle of sliding window analysis is fairly simple—a DNA alignment is divided up into multiple smaller fragments throughout the length of the region in question. Measures of diversity (Roe & Sperling, 2007) or questions of phylogenetic character conflict (Cruickshank, 2011) can then be explored.

The SPIDER function `slidingWindow()` provides the base for conducting sliding window analyses.

```
anoWin <- slidingWindow(anoteropsis, width=50, interval=1)
length(anoWin)
```

```
[1] 359
```

`slidingWindow()` takes an alignment and slices it into pieces of `width` bp each, each piece separated by `interval` bp from the one before it. The result is a big list of `DNAbin` objects ready and waiting to be analysed, as can be seen here:

```
anoWin[[1]]
```

```
33 DNA sequences in binary format stored in a matrix.

All sequences of same length: 50

Labels: Artoria_flavimanus Artoria_separata Anoteropsis_adumbrata_CO
Anoteropsis_adumbrata_BP Anoteropsis_aerescens_MK Anoteropsis_aerescens_TK ...

Base composition:
    a     c     g     t
0.264 0.142 0.137 0.458
```

5.2 Conducting analyses

If, for example, we wanted to see the change in GC content across the `anoteropsis` sequence, we can `sapply()` the APE command `GC.content()` across our windows:

```
anoGC <- sapply(anoWin, GC.content)
head(anoGC)
```

```
[1] 0.2787879 0.2775758 0.2575758 0.2575758 0.2763636 0.2703030
```

If we plot the results of `anoGC`, we get Figure 2.

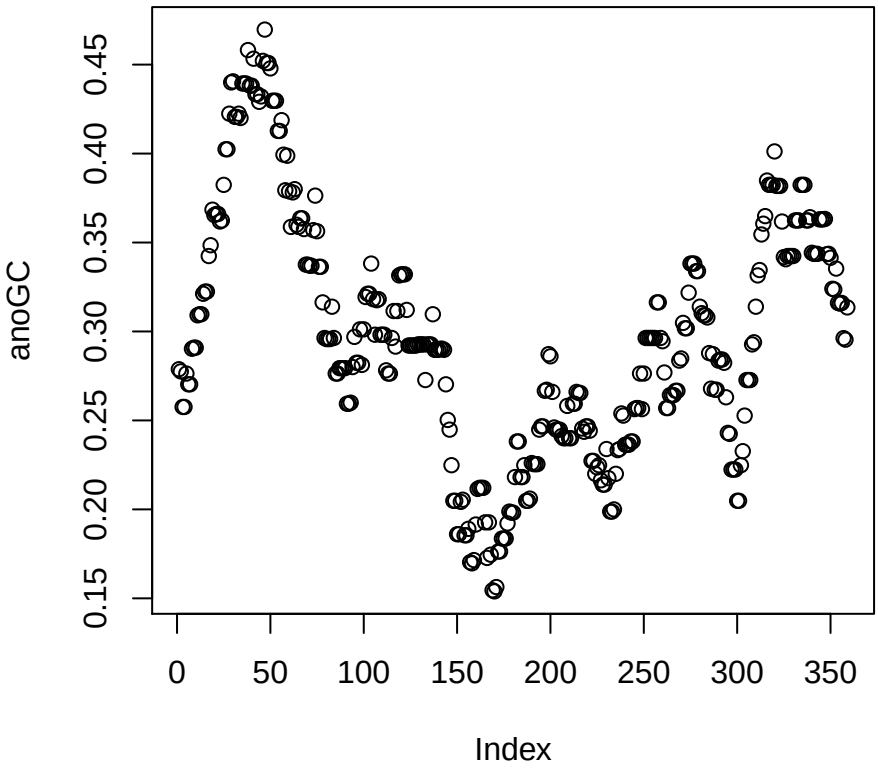

Figure 2: Average GC content of 50 bp windows across *Anoteropsis* spp. sequences

5.3 The magic of `slideAnalyses`

A number of the most useful analyses for determining the optimum region for designing mini-barcodes (Meusnier et al., 2008) have been implemented in the `slideAnalyses()` function. This function does the whole shebang from creating the windows, to conducting the analyses. The result is a massive list detailing the data from each window. Take note that this is our first glimpse of 'species vectors' in action:

```
anoAna <- slideAnalyses(anoteropsis, 50, anoSpp, interval="codons",
distMeasures=TRUE, treeMeasures=TRUE)
```

First up: the arguments. `anoteropsis` is our `DNAbin` object containing our sequences, '50' is the desired width of the windows in question. `anoSpp` is our species vector, `"codons"` is a fancy way of getting an interval of 3 between our windows, `distMeasures=TRUE` indicates that we want to calculate metrics based on a Kimura 2-parameter distances matrix for each window, and `treeMeasures=TRUE` indicates that we want to calculate metrics based on a neighbour-joining tree from the distance matrix above. The default is that `distMeasures=TRUE` and `treeMeasures=FALSE` because the latter takes a lot more time to compute than the former.

If we have a look at the object created by the function above, we get something fairly nasty:

```
str(anoAna)
```

```
List of 16
 $ win_mono_out   : num [1:120] 0.864 0.864 0.909 0.909 0.864 ...
 $ comp_depth_out : num [1:120] 0.588 0.471 0.529 0.529 0.412 ...
 $ comp_out       : num [1:120] 0.355 0.29 0.323 0.323 0.258 ...
 $ pos_tr_out     : num [1:120] 1 4 7 10 13 16 19 22 25 28 ...
 $ noncon_out     : num [1:120] 0.333 0.394 0.212 0.212 0.303 ...
 $ thres_below_out: num [1:120] 409 404 369 357 374 411 370 384 402 402 ...
 $ thres_above_out: num [1:120] 5 4 11 10 24 18 24 21 7 7 ...
 $ zero_out       : num [1:120] 0.0303 0.0398 0.0265 0.0246 0.036 ...
 $ dist_mean_out  : num [1:120] 0.0759 0.0758 0.0822 0.0848 0.0814 ...
 $ pos_out        : num [1:120] 1 4 7 10 13 16 19 22 25 28 ...
 $ dat_zero_out   : num 0
 $ boxplot_out    : logi FALSE
 $ distMeasures   : logi TRUE
 $ thresA         : num 0.2
 $ thresB         : num 0.1
 $ treeMeasures   : logi TRUE
 - attr(*, "class")= chr "slidWin"
```

win_mono_out is the proportion of species that are monophyletic. The closer to 1 this number gets, the more species are monophyletic. comp_depth and comp_depth_out compare the clades of the trees in the window to the trees made from the full dataset. The difference is that comp_depth_out considers only the clades that are closer to the tips of the tree than the median node depth, while comp_out looks at all of them. pos_out is the position of each window (in bp) from the start of the sequence. Occasionally, trees might not be able to be made if there is too much missing data, and pos_tr_out records the position of windows from which trees were made. noncon_out gives the proportion of non-conspecific distances that are 0 i.e. no difference between species. Ideally, this number would be 0. thres_above_out and thres_below_out are the numbers of cells with distances above and below given thresholds (thresA and thresB). It is best if these numbers are maximised and minimised respectively. zero_out is the proportion of cells in the distance matrix which have a distance of 0, and can be compared with dat_zero_out which is the same proportion in the full dataset. The closer zero_out is to dat_zero_out, the better. dist_mean_out is the mean of the distance matrix. Usually, it is best if this number is large. Finally, the last five objects in the list are used by plot() in creating the plots described below.

All the information above can be presented graphically using the plot command to produce Figure 3:

```
plot(anoAna)
```

Starting at the top left of Figure 3, we have the plot of the mean distance (dist_mean_out). We see that this is at its greatest at the 200 bp position, and at its lowest around 150 bp. Next one down the column, we have the proportion of zero cells in the distance matrix (zero_out). Once again, this is minimised around 200 bp and is at its highest at 150 bp. The unbroken horizontal line crossing the y-axis at 0 is the proportion of zero cells in the distance matrix created from the full dataset (dat_zero_out). At the bottom of the first column, we have the number of cells above and below the thresholds (thres_above_out and thres_below_out). The two metrics meet their maximum and minimum respectively at (surprise, surprise!) 200 bp. The top right plot displays the proportion of zero non-conspecific distances, which find their minimum at 200 bp, but is greatest (a scary 0.8) around 100 bp. The middle right plot shows the proportion of clades that are identical between the windows and the full dataset. Black nodes comprise all clades, while the blue ones are a subset making up the shallower clades. In this example they're mixed up with each other, due to the fact that this dataset has few representatives of each species. For datasets including lots of conspecific individuals, there is greater separation between the blue and the black points. As we've come to expect though, both measures are pretty high at 200 bp and low at 140 bp. Finally, at the bottom right we have the proportion of species that are monophyletic. Unlike previous measures, it appears that the 50 bp mark is best (though 200 bp is fairly respectable), but

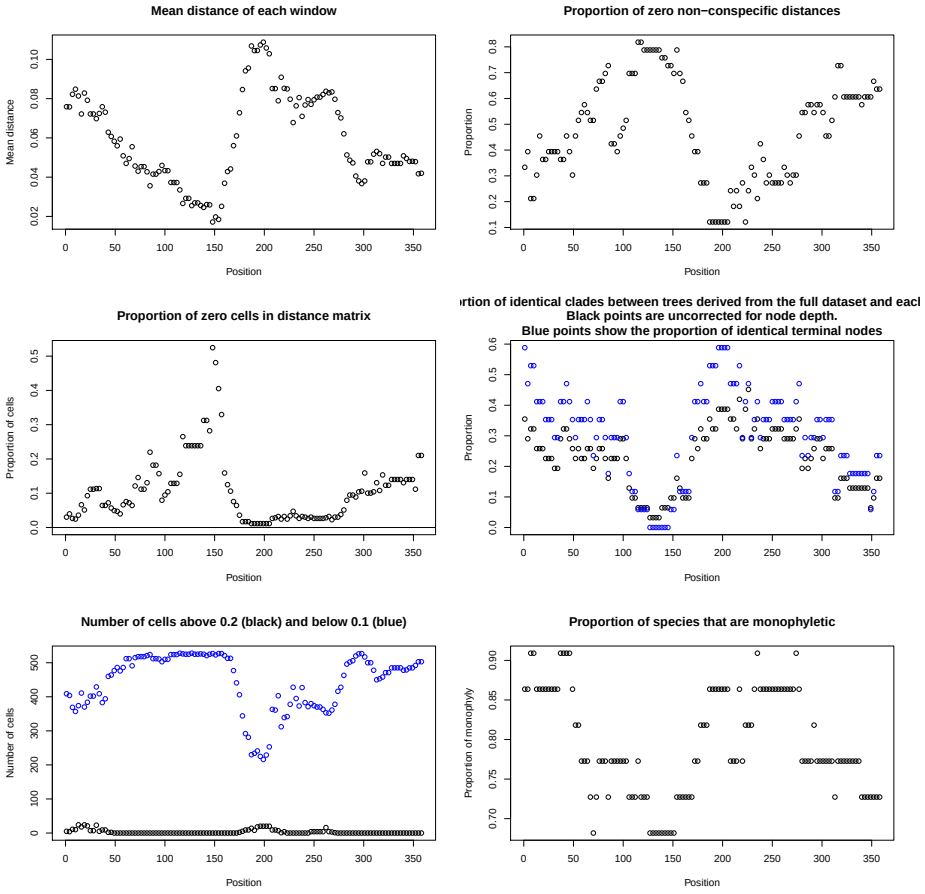

Figure 3: Results of several analyses across the COI sequences of *Anoteropsis* spp. See text for more details.

140 is still in the doldrums. All in all, these plots consistently indicate that the best region for creating 50 bp mini-barcodes is around the 200 bp mark.

The information can also be shown as a table that ranks the windows, showing by default the top 10 windows :

```
rankSlidWin(anoAna)
```

```
   position mean_distance monophyly clade_comparison clade_comp_shallow
67      199    0.10878089 0.8636364        0.3870968          0.5882353
66      196    0.10733553 0.8636364        0.3870968          0.5882353
68      202    0.10581874 0.8636364        0.3870968          0.5882353
69      205    0.10287159 0.8636364        0.3870968          0.5882353
63      187    0.10685304 0.8636364        0.3548387          0.5294118
65      193    0.10453572 0.8636364        0.3225806          0.5294118
64      190    0.10446778 0.8636364        0.3225806          0.5294118
4        10    0.08482881 0.9090909        0.3225806          0.5294118
3         7    0.08223324 0.9090909        0.3225806          0.5294118
73      217    0.09090806 0.8636364        0.4193548          0.5294118
   zero_noncon zero_distances above_threshold below_threshold
67   0.1212121     0.01136364              20              216
66   0.1212121     0.01136364              20              225
68   0.1212121     0.01136364              20              229
69   0.1212121     0.01136364              20              253
63   0.1212121     0.01136364              14              230
65   0.1212121     0.01136364              18              241
64   0.1212121     0.01136364               8              234
4    0.2121212     0.02462121              10              357
3    0.2121212     0.02651515              11              369
73   0.1818182     0.02462121               1              312
```

It shows the same conclusion that we drew from the graphical representation of the data, namely that the windows around the 200 mark are particularly good by most of the criteria.

5.4 Plots of boxes

Another way of looking at the variation across the sequence is by plotting boxplots that show the distribution of pairwise genetic distances of each window. This can be done by using the function `slideBoxplots()`. This function has three methods that be used. `"overall"` shows a single boxplot that represents the whole distance matrix, `"interAll"` separates the distances into intra- and inter-specific distances and plots a boxplot of each. This differs from `"nonCon"` (the default) in that `"nonCon"` restricts the inter-specific distances to only the nearest non-conspecific distance for each individual in the dataset.

```
anoBox <- slideBoxplots(anoteropsis, 50, anoSpp, interval="codons",
method="nonCon")
plot(anoBox)
```

When plotted, this function produces the graphic shown in Figure 4. Unfortunately, there is very little difference between the intra and inter-specific distances at the 50 bp window size. Even the windows around the 200 bp position show significant overlap between the two measures, showing that there is no 'barcoding gap' with this size of DNA fragment length.

To look at a more conventional example of the 'barcoding gap' we can have a look at the `weevils` dataset (Figure 5):

```
weevilsBox <- slideBoxplots(weevils, 50, weevilSpp1, interval="codons",
method="nonCon")
plot(weevilsBox)
```

6 Barcoding identification metrics

DNA barcoding is a method of identification of organisms that involves sequencing standard gene regions (particularly the mitochondrial gene cytochrome *c* oxidase I (COI)) and comparing these sequences to a database of sequences from identified specimens. The method was first proposed by Hebert et al. (2003) and has since become a global endeavour, with ongoing efforts to gather sequences and build the database, culminating in BOLD, which we encountered in Section 3.5. Research into the utility of DNA barcoding in different taxa centres on determining the amount of sequence variation within and between species and whether accurate determinations can be made.

6.1 Summary statistics

SPIDER provides two functions for determining summary statistics from your data. The first (`dataStat()`) calculates the number of species and genera in the dataset, while the second (`seqStat()`)determines sequence number and length.

```
doloGen <- rep("Dolomedes", length(doloSpp))
dataStat(doloSpp, doloGen)
```

```
 Genera Species     Min     Max  Median    Mean  Thresh
      1       4       2      16      10       9       1
```

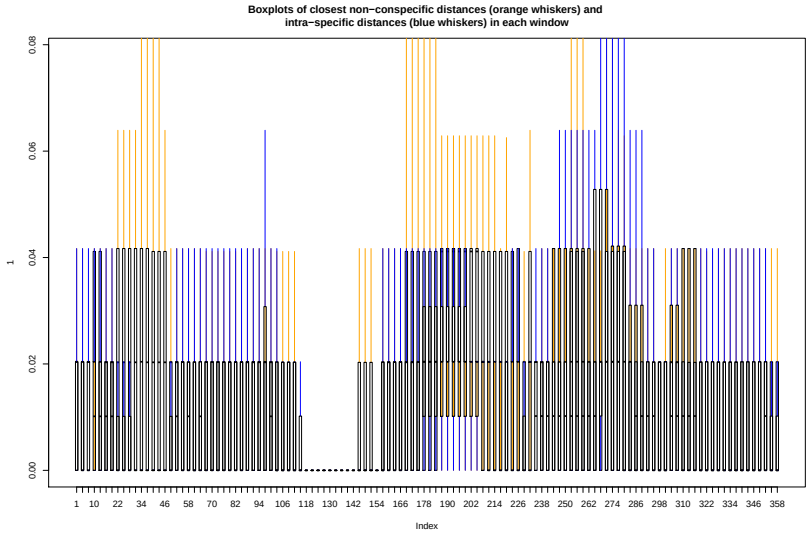

Figure 4: Boxplots showing the distribution of the nearest non-conspecific (orange) and intraspecific (blue) distances across the COI sequences of *Anoteropsis* spp.

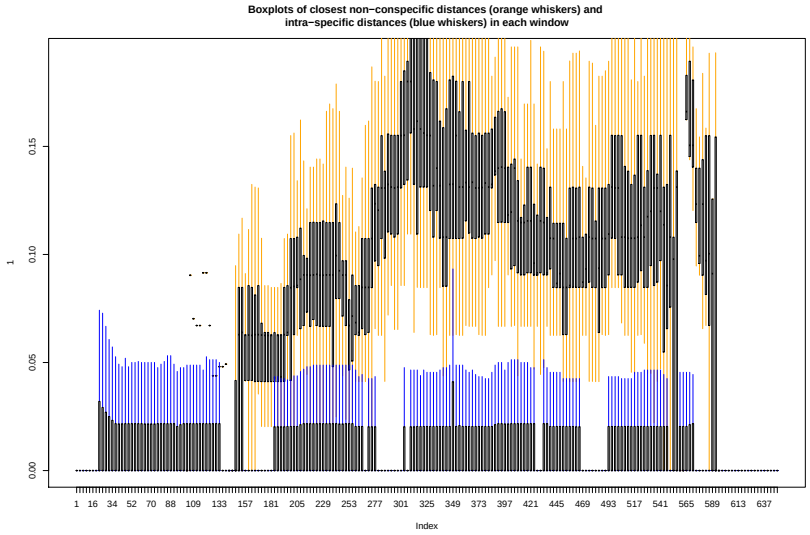

Figure 5: Boxplots showing the distribution of the nearest non-conspecific (orange) and intraspecific (blue) distances across the COI sequences of *Trigonopterus* spp.

The output is fairly easy to interpret. In the `dolomedes` dataset we have one genus and four species. The minimum number of individuals per species is two, while the maximum is 16. The median and mean number per species is 10 and 9 respectively. The "`thresh`" value shows how many species have fewer individuals than the threshold (default of 5). In the `dolomedes` dataset, we only have one species with less than 5 individuals representing it.

Looking at the `dolomedes` dataset with `seqStat()` is boring—all the sequences are the same length. Therefore, we will have a look at the `weevils` dataset:

```
seqStat(weevils)
```

```
   Min    Max   Mean Median Thresh
   479    646    635    640      1
```

The minimum sequence length is 479 bp, while the maximum is 646. The mean and median lengths are 635 and 640 respectively, and we have one sequence that is below the threshold value of 500 bp.

6.2 Distance measures of identification success

The following metrics are all based on the pairwise distance matrix between sequences. This matrix can easily be created using the `dist.dna()` function provided by APE. The distance matrix can be calculated using various models of DNA evolution, ranging from striaght raw distances to complex models involving a lot of parameters. The standard model of evolution used in DNA barcoding studies is the Kimura 2-parameter model (K2P), which is one of the more simple models available. Conveniently, this is the default model used by `dist.dna()`.

```
anoDist <- dist.dna(anoteropsis)
doloDist <- dist.dna(dolomedes)
```

6.2.1 Nearest neighbour

The nearest neighbour criterion (`nearNeighbour()` simply finds the closest individual to the target, and returns the species index for that individual. If there is more than one individual that is closest to the target, the function returns the species index of the major component of the group. The default result is a logical vector that tells you if the nearest species index is the same as the individual being tested. If it is, the result is TRUE. If not, it's FALSE. Setting the `names` argument to TRUE returns the name of the nearest match instead, and thus can be used to identify unknowns (subject to having tested the procedure previously of course!).

```
nearNeighbour(doloDist, doloSpp)
```

```
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
[37]  TRUE
```

6.2.2 BOLD identification criteria

The function `idBOLD()` mimics the "species identification" method used by BOLD. The process is as follows: two thresholds are given. If only one species is within the lower threshold, the function outputs TRUE. If more than one species is within that threshold, the function outputs FALSE. If there are no matches within the higher threshold, the result is NA. The default thresholds are 0.01 and 0.05 (1% and 5%).

```
idBOLD(anoDist, anoSpp)
```

```
 [1]    NA    NA FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
[13] FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE
[25] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE
```

6.2.3 Meier's best match

Meier's best close match (Meier et al., 2006) is the most conservative of the distance-based analyses, with a default threshold value of 0.01. If there are no matches within this threshold, a result of `"no id"` is given. If the nearest match is the same as the test individual, the result is `"correct"`, if the nearest match is different, it is `"incorrect"`. This test also gives us an extra degree of uncertainty, with a result of `"ambiguous"` if there is more than one species closest to the test individual.

```
meiersBestMatch(anoDist, anoSpp)
```

```
 [1] "no id"     "no id"     "no id"     "no id"     "incorrect" "no id"
 [7] "no id"     "no id"     "correct"   "correct"   "no id"     "no id"
[13] "no id"     "no id"     "incorrect" "no id"     "correct"   "no id"
[19] "no id"     "correct"   "no id"     "no id"     "correct"   "correct"
[25] "no id"     "no id"     "no id"     "no id"     "no id"     "correct"
[31] "correct"   "correct"   "no id"
```

6.3 Species monophyly

The species monophyly criterion determines if each species is monophyletic over the tree given in the function. It uses a fairly simple definition of monophyly—search for clades with the same number of tips as there are species members, and see if they match. Singletons (species represented by only one individual) cause a headache. Obviously, they are either always or never monophyletic depending on your point of view. The way that `monophyly()` gets around this problem is by getting the user to tell it what the point of view should be. The default is TRUE—singletons are monophyletic.

First we need to make our tree and ensure the labels are the same as the species vector. Then we can test for monophyly:

```
anoTr <- nj(anoDist)
anoTr$tip.label <- anoSpp
monophyly(anoTr)
```

```
 [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

The species are ordered in the same way as `unique(anoSpp)`, so this output tells us that *Anoteropsis adumbrata* is not monophyletic on this tree. However, if you've looked at the `anoteropsis` dataset at all, you'll know that there are a number of singletons in this dataset. When we change the default behaviour to consider singletons as FALSE, we get a rather different picture.

```
monophyly(anoTr, singletonsMono=FALSE)
```

```
 [1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
[13]  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE
```

6.4 Sensitivity analysis

Sensitivity analysis is a method for determining the optimum threshold for any given taxon (Meyer & Paulay, 2005). It does this by using the varying rates of false positives and false negatives when the identification accuracy is assessed at different thresholds. The function `idError()` provides the basis for this analysis.

```
idError(doloDist, doloSpp)
```

```
     Thresh Pos False neg False pos
[1,]   0.01  25        12         0
```

The function returns the number of positive identifications, the number of false negatives and the number of false positives at a given threshold. The threshold has a default of 0.01, but can be changed as the user pleases. If the threshold is changed systematically, the optimum threshold value can be determined. The following code shows how to create a range of threshold values, how to test these values, and how to plot the result (Figure 6):

```
threshVal <- seq(0.001,0.02, by = 0.001)
sens <- lapply(threshVal, function(x) idError(doloDist, doloSpp, thresh = x))
sensMat <- do.call(rbind, sens)
barplot(t(sensMat)[3:4,], names.arg=sensMat[,1])
```

From the figure we can see that the optimum threshold for the `dolomedes` dataset is between 0.004 and 0.007.

7 Conclusion

This tutorial gives an introduction the the usage of some of the functions implemented in SPIDER, particularly with regards to its sliding window capabilities and use in DNA barcoding research. SPIDER is an actively developing package, and it is intended that further analyses will be developed within it, with an especial focus on the analysis of conflict in phylogenetic data, checks of alignment quality, tools for studying ancient DNA and further analyses for morphological and categorical data. As the package matures, it is hoped that SPIDER will be a valuable addition to toolkit of R packages available for use in the field of taxonomy, systematics and molecular biology. For more information, or to get involved in the development of the package, please visit our site on R-Forge, http://spider.r-forge.r-project.org/.

References

Cruickshank, R. H. (2011). Exploring character conflict in molecular data. *Zootaxa*, *2946*, 45–51.

Greenslade, P., Stevens, M. I., Torricelli, G., & D'Haese, C. A. (2011). An ancient Antarctic endemic genus restored: morphological and molecular support for *Gomphiocephalus hodgsoni* (Collembola: Hypogastruridae). *Systematic Entomology*, *36*(2), 223–240.

Guoy, M., Guindon, S., & Gascuel, O. (2010). SeaView version 4: a multiplatform graphical user interface for sequence alignment and phylogenetic tree building. *Molecular Biology and Evolution*, *27*(2), 221–224.

Hebert, P. D. N., Cywinska, A., Ball, S. L., & de Waard, J. R. (2003). Biological identification through DNA barcodes. *Proceedings of the Royal Society of London. B.*, *270*, 313–321.

Jombart, T. (2008). adegenet: a R package for the multivariate analysis of genetic markers. *Bioinformatics*, *24*, 1403–1405.

Meier, R., Shiyang, K., Vaidya, G., & Ng, P. (2006). DNA barcoding and taxonomy in Diptera: a tale of high intraspecific variability and low identification success. *Systematic Biology*, *55*(5), 715–728.
URL http://sysbio.oxfordjournals.org/content/55/5/715.abstract

Meusnier, I., Singer, G. A. C., Landry, J.-F., Hickey, D. A., Hebert, P. D. N., & Hajibabaei, M. (2008). A universal DNA mini-barcode for biodiversity analysis. *BMC Genomics*, *9*(214), 1–4.

Meyer, C. P., & Paulay, G. (2005). DNA barcoding: error rates based on comprehensive sampling. *PLoS Biology*, *3*(12), 2229–2238.

Paradis, E. (2010). pegas: an R package for population genetics with an integrated–modular approach. *Bioinformatics*, *26*, 419–420.

Paradis, E., Claude, J., & Strimmer, K. (2004). APE: analyses of phylogenetics and evolution in R language. *Bioinformatics*, *20*, 289–290.

Riedel, A. (2010). One of a thousand—a new species of *Trigonopterus* (coleoptera, curculionidae, cryptorhynchinae) from New Guinea. *Zootaxa*, *2403*, 59–68.

Riedel, A., Daawia, D., & Balke, M. (2010). Deep cox1 divergence and hyperdiversity of *Trigonopterus* weevils in a New Guinea mountain range (Coleoptera, Curculionidae). *Zoologica Scripta*, *39*(1), 63–74.

Roe, A. D., & Sperling, F. A. H. (2007). Patterns of evolution of mitochondrial cytochrome *c* oxidase I and II DNA and implications for DNA barcoding. *Molecular Phylogenetics and Evolution*, *44*, 325–345.

Tamura, K., Peterson, D., Peterson, N., Stecher, G., Nei, M., & Kumar, S. (2011). MEGA5: Molecular Evolutionary Genetics Analysis using maximum likelihood, evolutionary distance, and maximum parsiomony methods. *Molecular Biology and Evolution*, *In Press*.

Venables, W. N., & Ripley, B. D. (2002). *Modern Applied Statistics with S*. New York: Springer, fourth ed. ISBN 0-387-95457-0.
URL http://www.stats.ox.ac.uk/pub/MASS4

Vink, C. J., & Dupérré, N. (2010). Pisauridae (Arachnida: Araneae). *Fauna of New Zealand*, *64*, 1–54.

Vink, C. J., & Paterson, A. M. (2003). Combined molecular and morphological phylogenetic analyses of the New Zealand wolf spider genus *Anoteropsis* (Araneae: Lycosidae). *Molecular Phylogenetics and Evolution*, *28*, 576–587.

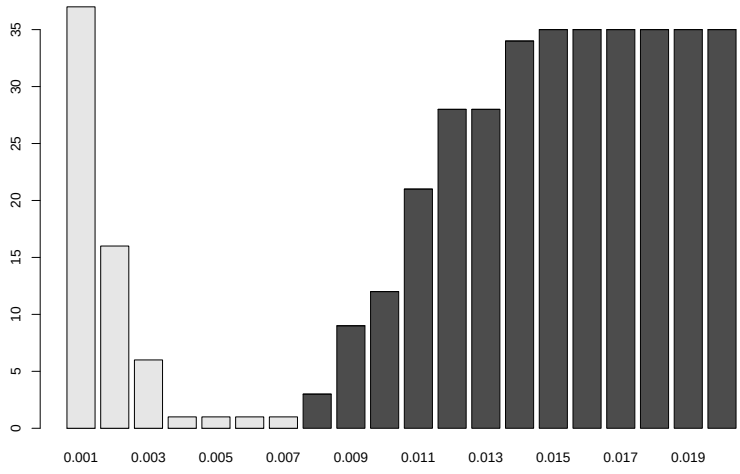

Figure 6: Barplots showing the false positive (light grey) and false negative (dark grey) rate of identification of *Dolomedes* species as the threshold changes.