**Preparing the data:**

1) cut recording in half, creating a training set and test set

2) Next to each set, create a column that identifies 10 second gaps in the recording. For ease of identification, I make the cell next to each 10 second gap say something big like “GAP”, and all other cells say something small like “.”

3) Copy and paste the sequence such that n-grams are shown in adjacent columns (this is done by copy/pasting the sequence one cell higher in each adjacent column). Do this for every n-gram from 1 to 7. So there will be 7 sets of columns: the first set will have only the original column; the second set will have two columns representing bigrams; …the 7th set will have 7 columns, with each column set one row higher such that the 7-grams read from left to right on each row. It is important that the sets are all lined up on the same row as the 10 second gap identification. (see the ‘combined’ tab from “BBF11 Sequence Luke.xls”. (see the ‘train’ and ‘test’ tabs from “BBF11 Sequence Luke.xls”)

4) Remove all n-grams that contain a ten-second gap. For each ten-second gap there will be one more n-gram deleted for each higher rank of n-gram: for example, each gap appears in two trigrams but only one bigram.

The easiest way I found to do these deletions was to scroll down, scanning with eyes, and at each “GAP” select the n-grams that contain that gap. Once all n-grams in all sets have been selected, I would delete them all at once (with ‘shift up’ selected). The sets will no longer be lined up with the original, but they should each be the list of n-grams that does not contain any ten-second gaps (see the ‘train’ and ‘test’ tabs from “BBF11 Sequence Luke.xls”).

5) There should now be lists of 1 to 7-grams for both the training set and test set. Each of these lists should be saved as its own separate .csv file. If doing a lot of these, (especially on the Fisher’s test side of the study), it is worth knowing that this copy/paste to a new file can be done very quickly with keyboard shortcuts. On a mac the following sequence can be typed on the keyboard within a couple of seconds: Command+C (copy), Command+N (new file), Command+V (paste), Command+Shift+S (save as), type name of file, Enter.

**Generating Simulations:**

6) To generate 0-order simulations you need the proportions that each song occurs in the training sequence (pivot table can get you that). In order to maintain the rule of immediate variety, I made an adjustment: I essentially transferred these proportions into a 1st-order table, but adjusted each row so that any identical bigrams pairs were set to 0. So for example, in this table there will be a row of proportions for songs that follow song A in the first row of the table. These proportions will be based on the proportions of each song in the sequence regardless of what actually came after song A in the sequence. However, these need to be adjusted: the proportion of A-A needs to be 0 (for immediate variety), so whatever proportion was in the A-A cell needs to be redistributed to the other songs. This can be achieved using Macros in excel and an understanding that the sum of each row of probabilities needs to equal 1. So if A-A was 0.2, then the sum for the whole row will now equal 0.8 once A-A is set to 0. All the remaining proportions need to be multiplied by 1/0.8 in order to redistribute the proportions so that the row once again sums to 1.

Similarly the 2nd row will have the same original proportions from the training set, but B-B will be set to 0 and redistributed among the remaining proportions in that row. So if B-B occurred 0.1, then it will now be 0, and all remaining proportions in that row will need to be multiplied by 1/0.9. This adjustment needs to be made for each row of the table but can be accomplished quite quickly with the right Macros (see ‘0 order’ tab of ‘BBF11 sequence Luke.xls’)

7) Copy/paste the resulting table into a new file and save the table as a .csv file (see ‘zerotable.csv’).

8) The following code can be used to generate 1000 0-order simulations that are 612 songs long, using a table called ‘zerotable.csv’ (612 even though it says 611 in the code, because it adds a final song at the 611th song). It then creates a file with these simulations called ‘zerosims.csv’. This first half of the code generates an initial sequence that starts the data frame. The second half of the code adds 1000 sequences to that data frame before printing it out.

zerotable = read.csv("zerotable.csv", sep=",",row.names=1)

Sequence9 = 'F'     //create Sequence9 and put 'A' in it

**for(i in 1:611)**

**{prev=Sequence9[i]**

**Sequence9[i+1] = sample(colnames(zerotable), size=1, prob=zerotable[prev,])};**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

tumbas <- data.frame(Sequence9)

for(j in 1:1000){

**for(i in 1:611)**

**{prev=Sequence9[i]**

**Sequence9[i+1] = sample(colnames(zerotable), size=1, prob=zerotable[prev,])};**

tumbas <-cbind(tumbas,Sequence9)}

write.csv(tumbas, file = 'zerosims.csv')

**Generating 1st Order Simulations**

9) (see 1 order tab from ‘BBF11 Sequence Luke’) 1st order simulations are generated almost exactly the same as the above. The only difference is that you use a transition table that actually corresponds to 1st order information instead of the adjusted overall proportions used above (steps 6-8).

So generate a transition table using the bigram columns from the training set. Be sure to set the values to ‘proportion of each row’ so it reflects the proportions of songs that follow each individual song rather than just the total proportion of songs. Save that table as a .csv. Then use the above code from step 8, only with different names for the input and output so as to distinguish it from the 0-order simulations.

**Generating 2nd Order Simulations:**

10) 2nd order simulations is a little bit trickier in that excel offers pivot tables that have unwanted summed rows within them. So generate a pivot table using the trigram training set columns. 1st and 2nd columns should be set to the ‘rows’ of the pivot table, and 3rd columns should be set to the ‘columns’ part of the pivot table. I then copy/pasted the values from that table to a separate space so I could manipulate it. I deleted the summary rows and set the 1st column to a list of all possible bigrams. This list can be generated in R using the gtools package and the following code that would create a data.frame with all the bigrams of the first 11 letters of the alphabet listed in the first column. This would be for a bird that had an 11-song repertoire.

twograms <- permutations(11,2,LETTERS[1:11],repeats=TRUE)

twogramtemp <- apply(twograms, 1, function(x)paste0(x, collapse=''))

twogramdf <- data.frame(twogramtemp)

twogramdf<- data.frame(twogramdf, row.names = 1)

twogramdf[,1]=0

After generating, this list could be printed into excel using the write.csv command and then copy/pasted into the table. At the beginning of each bird’s calculation I simply generated all such permutation dataframes in R so that they could be called upon at any time. Here is the entire list of code for all permutation lists from 1 to 7-grams (only for a bird with an 11-song repertoire. For other repertoires simply use find/replace to change all ‘11’ to the appropriate repertoire size).

onegrams <- permutations(11,1,LETTERS[1:11],repeats=TRUE)

onegramtemp <- apply(onegrams, 1, function(x)paste0(x, collapse=''))

onegramdf <- data.frame(onegramtemp)

onegramdf<- data.frame(onegramdf, row.names = 1)

onegramdf[,1]=0

twograms <- permutations(11,2,LETTERS[1:11],repeats=TRUE)

twogramtemp <- apply(twograms, 1, function(x)paste0(x, collapse=''))

twogramdf <- data.frame(twogramtemp)

twogramdf<- data.frame(twogramdf, row.names = 1)

twogramdf[,1]=0

threegrams <- permutations(11,3,LETTERS[1:11],repeats=TRUE)

threegramtemp <- apply(threegrams, 1, function(x)paste0(x, collapse=''))

threegramdf <- data.frame(threegramtemp)

threegramdf<- data.frame(threegramdf, row.names = 1)

threegramdf[,1]=0

fourgrams <- permutations(11,4,LETTERS[1:11],repeats=TRUE)

fourgramtemp <- apply(fourgrams, 1, function(x)paste0(x, collapse=''))

fourgramdf <- data.frame(fourgramtemp)

fourgramdf<- data.frame(fourgramdf, row.names = 1)

fourgramdf[,1]=0

fivegrams <- permutations(11,5,LETTERS[1:11],repeats=TRUE)

fivegramtemp <- apply(fivegrams, 1, function(x)paste0(x, collapse=''))

fivegramdf <- data.frame(fivegramtemp)

fivegramdf<- data.frame(fivegramdf, row.names = 1)

fivegramdf[,1]=0

sixgrams <- permutations(11,6,LETTERS[1:11],repeats=TRUE)

sixgramtemp <- apply(sixgrams, 1, function(x)paste0(x, collapse=''))

sixgramdf <- data.frame(sixgramtemp)

sixgramdf<- data.frame(sixgramdf, row.names = 1)

sixgramdf[,1]=0

sevengrams <- permutations(11,7,LETTERS[1:11],repeats=TRUE)

sevengramtemp <- apply(sevengrams, 1, function(x)paste0(x, collapse=''))

sevengramdf <- data.frame(sevengramtemp)

sevengramdf<- data.frame(sevengramdf, row.names = 1)

sevengramdf[,1]=0

11) The simulations can then be generated using the following code, similar to the 1st order code but combining the previous two songs to call upon the 2nd-order table proportions. The table is called BBF112ordertrainer.csv. The sequence is first ‘seeded’ with ‘F’ and ‘D’, the two songs that began the training set for BBF11. This sequence again happens to be 612 songs long.

BBF11twotrainer = read.csv("BBF112ordertrainer.csv", sep=",",row.names=1)

Sequence9='F'

Sequence9[2]='D'

for(i in 2:611) {

prevone = Sequence9[i]

prevtwo = Sequence9[i-1]

comboprev = paste(prevtwo, prevone, sep = '')

Sequence9[i+1] = sample(colnames(BBF11twotrainer), size=1, prob=BBF11twotrainer[comboprev,])}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

tumbas <- data.frame(Sequence9)

for(j in 1:1000){

for(i in 2:611) {

prevone = Sequence9[i]

prevtwo = Sequence9[i-1]

comboprev = paste(prevtwo, prevone, sep = '')

Sequence9[i+1] = sample(colnames(BBF11twotrainer), size=1, prob=BBF11twotrainer[comboprev,])};

tumbas<-cbind(tumbas,Sequence9)}

write.csv(tumbas, file = 'BBF112ordsims.csv')

**Generating 3rd-order simulations**

12) This is more challenging because the pivot tables generated by excel for tetragrams are quite large and unwieldy, and difficult to delete all the summary rows. So I used R instead of excel to give a proportion of each tetragram. I first generate an empty table in Excel that lists all possible trigrams on the left (using the permutation code from above, only adjusted to give trigrams), each song across the top, and filled with 0s. This is called “threegramtable.csv” in the code below. “BBF11fourtrain.csv” is simply the tetragram columns from the training set for BBF11 completed at step 5 above.

Here is the code used to count each tetragram and tally it in fourgramtraindf and print that as 4ordertrainer.csv.

fourgramtrain=read.csv("threegramtable.csv", sep=",")

fourgramtraindf <- data.frame(fourgramtrain, row.names = 1)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fourgrams=read.csv("BBF11fourtrain.csv", sep=",")

for(k in 1:603){

i = paste0(fourgrams[k,1], fourgrams[k,2], fourgrams[k,3])

j = paste0(fourgrams[k,4])

fourgramtraindf[i,j] = fourgramtraindf[i,j] + 1}

write.csv(fourgramtraindf, file = '4ordertrainer.csv')

Unfortunately that code only tallies instances of each tetragram rather than giving proportions (I’m sure it could be augmented to give proportions, but I didn’t bother, being quite new to R and tired of googling different error messages). I then used Excel Macros to turn those tallies into proportions for each row. The resulting table (see ‘BBF113ordertrainer.csv’) was used in the following code to generate 3rd order simulations. It is very similar to the code used for 2nd order simulations, only it combines the previous three songs rather than two.

BBF11threetrainer = read.csv("BBF113ordertrainer.csv", sep=",",row.names=1)

Sequence9='F'

Sequence9[2]='D'

Sequence9[3]='A'

for(i in 3:611) {

prevone = Sequence9[i]

prevtwo = Sequence9[i-1]

prevthree = Sequence9[i-2]

comboprev = paste(prevthree, prevtwo, prevone, sep = '')

Sequence9[i+1] = sample(colnames(BBF11threetrainer), size=1, prob=BBF11threetrainer[comboprev,])}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

tumbas <- data.frame(Sequence9)

for(j in 1:1000){

for(i in 3:611) {

prevone = Sequence9[i]

prevtwo = Sequence9[i-1]

prevthree = Sequence9[i-2]

comboprev = paste(prevthree, prevtwo, prevone, sep = '')

Sequence9[i+1] = sample(colnames(BBF11threetrainer), size=1, prob=BBF11threetrainer[comboprev,])};

tumbas<-cbind(tumbas,Sequence9)}

write.csv(tumbas, file = 'BBF113ordsims.csv')

13) 4th-order simulations can be generated just like the 3rd-order simulations, only with the appropriate changes to generate the table. Here is the code for generating the simulations once the table is constructed:

fourordtrainer=read.csv("fourordtrainer.csv", sep=",")

fourordtrainerdf <- data.frame(fourordtrainer, row.names = 1) //makes the left column into names rather than values.

Sequence9='F'

Sequence9[2]='D'

Sequence9[3]='A'

Sequence9[4]='H'

for(i in 4:611) {

prevone = Sequence9[i]

prevtwo = Sequence9[i-1]

prevthree = Sequence9[i-2]

prevfour = Sequence9[i-3]

comboprev = paste(prevfour, prevthree, prevtwo, prevone, sep = '')

Sequence9[i+1] = sample(colnames(fourordtrainerdf), size=1, prob=fourordtrainerdf[comboprev,])}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

tumbas <- data.frame(Sequence9)

for(j in 1:1000){

for(i in 4:611) {

prevone = Sequence9[i]

prevtwo = Sequence9[i-1]

prevthree = Sequence9[i-2]

prevfour = Sequence9[i-3]

comboprev = paste(prevfour, prevthree, prevtwo, prevone, sep = '')

Sequence9[i+1] = sample(colnames(fourordtrainerdf), size=1, prob=fourordtrainerdf[comboprev,])}

tumbas<-cbind(tumbas,Sequence9)}

write.csv(tumbas, file = 'fourordsims.csv')

**Calculating L Distances:**

14) LDistances can now be calculated using the simulation files and the test-set files. The following code will calculate LDistances of monograms for BBF11. I’ve added comments hopefully to explain it better. Onegramdf is simply a dataframe with the left column representing all the permutations of monograms generated from code similar to that in step 10; in the case of monograms, that is simply each song listed in the left column. For higher n-gram levels, this is more relevant because you would need all permutations of a given n-gram.

BBF11monotest = read.csv("BBF11monotest.csv", sep=",")

testsettemp <- onegramdf

for(k in 1:612){

j = paste0(BBF11monotest[k,1]);

testsettemp[j,1] = testsettemp[j,1] + 1}

testsettemp[,1] = testsettemp[,1]/612

Ldistance = 0

for(i in 1:1000)

{Simtemp <- onegramdf

Sequence <- tumbas[,i]

for(k in 1:**612**){

j = paste0(Sequence[k]);

Simtemp[j,1] = Simtemp[j,1] + 1;

}

div <- Simtemp[,1]/**612**

Ldistance[i] = sum(abs(testsettemp[,1]-div))

}

write.csv(Ldistance, file = 'monogramLdistance.csv')

//See ‘monogramLdistance.csv’ for the output.

Below is the code for calculating the Ldistances at the 7-gram level (with an added bit of code for calculating how long the code took to execute). Note the numbers corresponding to sequence length have changed because the ‘BBF11seventest.csv” is a shorter sequence than ‘BBF11monotest.csv’, due to more 10-second gaps (each gap occurs in 6 different 7-grams). Also, there are fewer 7-grams simply because the last 7-gram in the sequence begins on the 7th-last song whereas the final monogram occurs on the last song. Unfortunately having the numbers off in the code even by one can cause an error. It can’t

Note that this code only measures one simulated sequence rather than a thousand, due to time constraints. The output, ‘sevengramLdistance.csv’ correspondingly has only one value.

Hopefully by comparing the monogram and 7-gram code someone can learn how to adjust the code for all the different n-gram levels. The key is adjusting the sequence lengths because these are different for every test set. I would have a tally on my desktop of how long each test and training set were, from monograms to 7-grams.

BBF11seventest=read.csv("BBF11seventest.csv", sep=",")

testsettemp <- sevengramdf

for(k in 1:600){

j = paste0(BBF11seventest[k,1], BBF11seventest[k,2], BBF11seventest[k,3], BBF11seventest[k,4], BBF11seventest[k,5], BBF11seventest[k,6], BBF11seventest[k,7]);

testsettemp[j,1] = testsettemp[j,1] + 1}

testsettemp[,1] = testsettemp[,1]/600

start\_time <- Sys.time()

Ldistance = 0

for(i in 1:1)

{Simtemp <- sevengramdf

Sequence <- tumbas[,i]

for(k in 1:606){

j = paste0(Sequence[k], Sequence[k+1], Sequence[k+2], Sequence[k+3], Sequence[k+4], Sequence[k+5], Sequence[k+6]);

Simtemp[j,1] = Simtemp[j,1] + 1;

}

div <- Simtemp[,1]/606

Ldistance[i] = sum(abs(testsettemp[,1]-div))

}

write.csv(Ldistance, file = 'sevengramLdistance.csv')

end\_time <- Sys.time()

end\_time - start\_time

15) Feel free to email [lukecmclean@gmail.com](mailto:lukecmclean@gmail.com) with any questions!