Appendix - Selection against instability: stable subgraphs are most frequent in empirical food webs

Jonathan J. Borrelli

Wednesday, July 23, 2014

Contents

Borrelli, J. J. 2014. Selection against instability: stable subgraphs are most frequent in empirical food webs. -Oikos 000: 000-000

The code provided here can also be obtained from GitHub by clicking this link

Definitions

Subgraph Library

The following code defines the sign matrix for each of the thirteen possible three-node subgraphs. Here, the "s" in the object name indicates that only single links are used, while a "d" indicates the presence of double links.

```
s1<-matrix(c(0,1,0,-1,0,1,0,-1,0),nrow=3,ncol=3)
s2<-matrix(c(0,1,1,-1,0,1,-1,-1,0),nrow=3,ncol=3)
s3<-matrix(c(0,1,-1,-1,0,1,1,-1,0),nrow=3,ncol=3)
s4<-matrix(c(0,1,1,-1,0,0,-1,0,0),nrow=3,ncol=3)
s5<-matrix(c(0,0,1,0,0,1,-1,-1,0),nrow=3,ncol=3)

d1<-matrix(c(0,1,1,-1,0,1,-1,1,0),nrow=3,ncol=3)
d2<-matrix(c(0,1,1,1,0,1,-1,-1,0),nrow=3,ncol=3)
d3<-matrix(c(0,1,-1,1,0,0,1,0,0),nrow=3,ncol=3)
d4<-matrix(c(0,1,1,-1,0,0,1,0,0),nrow=3,ncol=3)
d5<-matrix(c(0,1,1,-1,0,1,1,-1,0),nrow=3,ncol=3)
d6<-matrix(c(0,1,1,1,0,1,1,1,0),nrow=3,ncol=3)
```

```
d7<-matrix(c(0,1,1,1,0,1,1,-1,0),nrow=3,ncol=3)
d8<-matrix(c(0,1,1,1,0,0,1,0,0),nrow=3,ncol=3)

mot.lst <- list(s1, s2, s3, s4, s5, d1, d2, d3, d4, d5, d6, d7, d8)
names(mot.lst) <- c("s1", "s2", "s3", "s4", "s5", "d1", "d2", "d3", "d4", "d5", "d6", "d7", "d8")
```

Define required functions

Functions for counting motifs

The `motif_counter` function takes in a list of graph objects and applies `triad.census` to each. It returns a data frame of the frequency of each connected three-node digraph.

```
motif_counter <- function(graph.lists){
  require(igraph)

  if(!is.list(graph.lists)){
    stop("The input should be a list of graph objects")
  }

  triad.count <- lapply(graph.lists, triad.census)
  triad.matrix <- matrix(unlist(triad.count), nrow = length(graph.lists), ncol = 16, byrow = T)
  colnames(triad.matrix) <- c("empty", "single", "mutual", "s5", "s4", "s1", "d4",
                              "d3", "s2", "s3","d8", "d2", "d1", "d5", "d7", "d6")

  triad.df <- as.data.frame(triad.matrix)

  motif.data.frame <- data.frame(s1 = triad.df$s1, s2 = triad.df$s2, s3 = triad.df$s3, s4 = triad.df$s4
                                 s5 = triad.df$s5, d1 = triad.df$d1, d2 = triad.df$d2, d3 = triad.df$d3
                                 d5 = triad.df$d5, d6 = triad.df$d6, d7 = triad.df$d7, d8 = triad.df$d8

  rownames(motif.data.frame) <- names(graph.lists)
  return(motif.data.frame)
}
```

The `permutes_rc` function is a null model that generates `iter` number of permuted matrices from the input matrix (`mat`). Each permuted matrix is created by randomly sampling 2x2 submatrices matching the pattern

```
     [,1] [,2]
[1,]    0    1
[2,]    1    0
```

or

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

and swapping the 0s and 1s. This null model preserves the number of prey and predators of each species (the degree distribution).

```
permutes_rc <- function(mat, iter = 100){

  pattern1 <- matrix(c(0,1,1,0), nrow = 2, ncol = 2)
  pattern2 <- matrix(c(1,0,0,1), nrow = 2, ncol = 2)
  count <- 0

  mat.list <- list()
  for(i in 1:iter){
    mat.list[[i]] <- matrix(0, nrow = nrow(mat), ncol = ncol(mat))
  }

  while(count < iter){
    srow <- sample(1:nrow(mat), 2)
    scol <- sample(1:ncol(mat), 2)

    test <- mat[srow, scol]

    if(sum(test == pattern1) == 4){
      count <- count + 1
      mat[srow, scol] <- pattern2
      mat.list[[count]] <- mat

      next
    } else if(sum(test == pattern2) == 4){
      count <- count + 1
      mat[srow, scol] <- pattern1
      mat.list[[count]] <- mat

      next
    } else {next}
  }

  matrices <- lapply(mat.list, as.matrix)
  return(permuted.matrices = matrices)
}
```

Functions for determining quasi sign-stability

There are two main functions for determining quasi sign-stability, and a third that wraps them together to generate the desired number of iterations.

The function `ran.unif` takes an input of a signed matrix. It will then check each cell to see if there is a 1 or -1. Each 1 will be replaced by a value drawn from the random uniform distribution between 0 and 10, while each -1 is replaced by a value from the random uniform distribution between -1 and 0. The `ran.unif` function also assigns values to the diagonal from a random uniform distribition between -1 and 0. The resulting randomly sample matrix is returned.

```
ran.unif <- function(motmat){
  newmat <- apply(motmat, c(1,2), function(x){
    if(x==1){runif(1, 0, 10)}else if(x==-1){runif(1, -1, 0)} else{0}
  })
  diag(newmat) <- runif(3, -1, 0)
  return(newmat)
```

```
}
```

Given the input matrix `maxRE` will compute the eigenvalues and return the largest real part.

```
maxRE <- function(rmat){
  lam.max <- max(Re(eigen(rmat)$values))
  return(lam.max)
}
```

The above two functions are combined in `eig.analysis`. Given the number of desired sampling iterations, `n`, and a list of sign matrices to analyze, `matrices`, the `eig.analysis` function will return an `n` by `length(matrices)` matrix of eigenvalues. Specifically it is returning the $max(Re(\lambda))$ for each sampled matrix. From this matrix quasi sign-stability can be calculated as the proportion of values in each column that are negative.

```
eig.analysis <- function(n, matrices){
  cols <- length(matrices)
  rows <- n
  eigenMATRIX <- matrix(0, nrow = rows, ncol = cols)
  for(i in 1:n){
    ranmat <- lapply(matrices, ran.unif)
    eigs <- sapply(ranmat, maxRE)
    eigenMATRIX[i,] <- eigs
  }
  return(eigenMATRIX)
}
```

Analysis

Load required packages

```
library(igraph)
library(ggplot2)
library(reshape2)
```

Determining motif frequency

Load in web data from GitHub. Click here to download the .Rdata file. This file is a list of igraph graph objects for each of the 50 webs used in the analysis. Once you have downloaded the file into your working directory:

```
load(paste(getwd(), "webGRAPHS.Rdata", sep = "/"))
```

The frequencies of each of the different subgraphs can now be determined easily with `motif_counter`.

```
motfreq <- motif_counter(web.graphs)
kable(motfreq, format = "pandoc")
```

	s1	s2	s3	s4	s5	d1	d2	d3	d4	d5	d6	d7	d8
akatorea	115	14	0	789	1466	0	0	0	0	0	0	0	0
akatoreb	63	18	0	338	497	0	0	0	0	0	0	0	0
benguela	269	445	0	464	391	19	48	8	24	0	1	1	0
berwick	132	14	0	855	1553	0	0	0	0	0	0	0	0
blackrock	407	71	0	2507	1976	0	0	0	0	0	0	0	0
bridgebrook	1931	629	0	6338	1539	30	111	0	21	0	4	0	0
broad	641	16	0	5087	4151	0	0	0	0	0	0	0	0
broom	527	358	0	275	3292	0	0	0	0	0	0	0	0
bsq	5742	1202	0	10011	5736	0	0	0	0	0	0	0	0
canton	717	116	0	6561	5976	0	0	0	0	0	0	0	0
catlins	59	9	0	186	721	0	0	0	0	0	0	0	0
caymen fw	22330	5965	0	64365	53833	0	0	0	0	0	0	0	0
chesapeake	86	21	0	58	130	0	0	0	0	0	0	0	0
coachella	311	638	0	424	317	103	139	96	91	11	24	39	7
coweeta1	68	36	0	266	551	0	0	0	0	0	0	0	0
coweeta17	86	26	0	421	822	0	0	0	0	0	0	0	0
csm	4540	944	0	5545	5327	0	0	0	0	0	0	0	0
cuban fw	23615	6220	0	65677	56327	0	0	0	0	0	0	0	0
dempstersau	539	28	0	3215	2255	0	0	0	0	0	0	0	0
dempsterssp	723	28	0	4054	4145	0	0	0	0	0	0	0	0
dempsterssu	2464	567	0	10208	8210	0	0	0	0	0	0	0	0
elverde	8571	2824	26	11485	10558	183	179	1170	2668	174	13	112	382
epb	10931	1730	0	16385	11214	0	0	0	0	0	0	0	0
flensburg	1653	571	0	3763	2712	0	0	0	0	0	0	0	0
german	494	87	0	1876	2057	0	0	0	0	0	0	0	0
grass	82	30	0	193	152	0	0	0	0	0	0	0	0
healy	1108	267	0	5119	4929	0	0	0	0	0	0	0	0
jamaican fw	24879	6989	0	73176	61667	0	0	0	0	0	0	0	0
kyeburn	657	213	0	6381	4470	0	0	0	0	0	0	0	0
lilkyeburn	654	35	0	2116	2026	0	0	0	0	0	0	0	0
littlerock	12210	9148	0	39239	11325	383	1039	63	1061	30	40	27	21
martins	498	114	0	1257	2195	0	0	0	0	0	0	0	0
narrowdale	83	27	0	357	1208	0	0	0	0	0	0	0	0
northcol	278	50	0	710	1731	0	0	0	0	0	0	0	0
otago	5920	1950	0	10779	9177	182	211	272	634	0	16	22	36
powder	270	38	0	945	1894	0	0	0	0	0	0	0	0
quick	2990	3059	2	6332	1873	419	1219	297	660	38	123	195	53
reef	1448	1694	9	2001	1103	147	219	147	354	56	17	55	16
shelf	5978	11211	2	11458	7760	130	132	33	79	13	0	4	1
skipwith	330	1169	0	1985	559	54	314	1	45	0	20	0	0
stmarks	617	223	0	442	569	0	0	0	0	0	0	0	0
stmartin	538	278	0	712	482	0	0	0	0	0	0	0	0
stony	1035	124	0	8929	8208	0	0	0	0	0	0	0	0
suttonau	206	16	0	2570	1554	0	0	0	0	0	0	0	0
suttonsp	220	4	0	3467	2067	0	0	0	0	0	0	0	0
suttonsu	312	6	0	6445	1728	0	0	0	0	0	0	0	0
sylt	4489	1654	0	6938	7989	45	67	50	75	0	0	1	2
troy	186	20	0	507	794	0	0	0	0	0	0	0	0
venlaw	186	17	0	638	932	0	0	0	0	0	0	0	0
ythan_nopar96	1247	362	0	1905	2551	13	4	21	8	0	0	0	0

The following code runs the null model analysis for the 50 food webs. First, each of the fifty webs are converted into binary adjacency matrices (`web.matrices`). Because the null model is a stochastic process the `set.seed(10)` allows for reproducible results. The code then loops through the list of adjacency matrices, generating 1000 permuted versions. The subgraphs are counted in each permuted matrix, and stored in a list (`p.mot`).

```
web.matrices <- lapply(web.graphs, get.adjacency, sparse = F)

set.seed(10)
pmot <- list()

for(i in 1:length(web.matrices)){
  p <- permutes_rc(web.matrices[[i]], 1000)
  g <- lapply(p, graph.adjacency)
  pmot[[i]] <- motif_counter(g)
  print(i)
}
```

Once subgraph counts have been obtained, the mean and standard deviation for each subgraph are computed. Z-scores are then computed as described in the methods section:

$$z_i = \frac{x_i - \overline{x}}{\sigma}$$

In cases where there were no occurences of a subgraph (standard deviation = 0) `NaN` is produced following the appllication of the above formula. In these cases I have replaced the `NaN` with 0. The normalized profile was then computed (as desribed in the methods):

$$n_i = \frac{z_i}{\sqrt{\sum z_j^2}}$$

```
mus <- t(sapply(pmot, function(x){colMeans(x)}))
sig <- t(sapply(pmot, function(x){apply(x, 2, sd)}))
```

```
z <- (motfreq - mus)/sig
zmat <- as.matrix(z)
zmat[is.nan(zmat)] <- 0
profile <- apply(zmat, 2, function(x){x/sqrt(rowSums(zmat^2))})
```

Figure 1a is then just a boxplot of the above object `profile`, reordered according to decreasing quasi sign-stability (see below).

Determining Quasi Sign-Stability

The first step to get quasi sign stability is to get the largest eigenvalues from a series of randomly parameterized sign matrices. In the following code I generate 10000 random parameterizations for each of the 13 subgraphs's sign matrices (`mot.lst`). The `eig.analysis` function will return a matrix where each column is a different subgraph and each row is the largest eigenvalue of a particular randomization.

```
set.seed(5)

n <- 10000
mot.stab<- eig.analysis(n, mot.lst)
colnames(mot.stab) <- names(mot.lst)
```

From that matrix, quasi sign-stability is calculated as the proportion of rows with a negative value. In other words, how many random parameterizations of the sign matrix were locally stable?

```
mot.qss <- apply(mot.stab, 2, function(x){sum(x<0)/n})
sorted <- sort(mot.qss, decreasing = T)
sorted
```

```
    s1     s4     s5     s2     d3     d4     s3     d2     d1     d5
1.0000 1.0000 1.0000 0.5345 0.0891 0.0866 0.0561 0.0428 0.0370 0.0101
    d7     d6     d8
0.0021 0.0000 0.0000
```

Correlation

Compute the correlation between QSS and median z-score

```
med.z <- apply(zmat[,names(sorted)], 2, median)

cor.test(sorted, med.z)
```

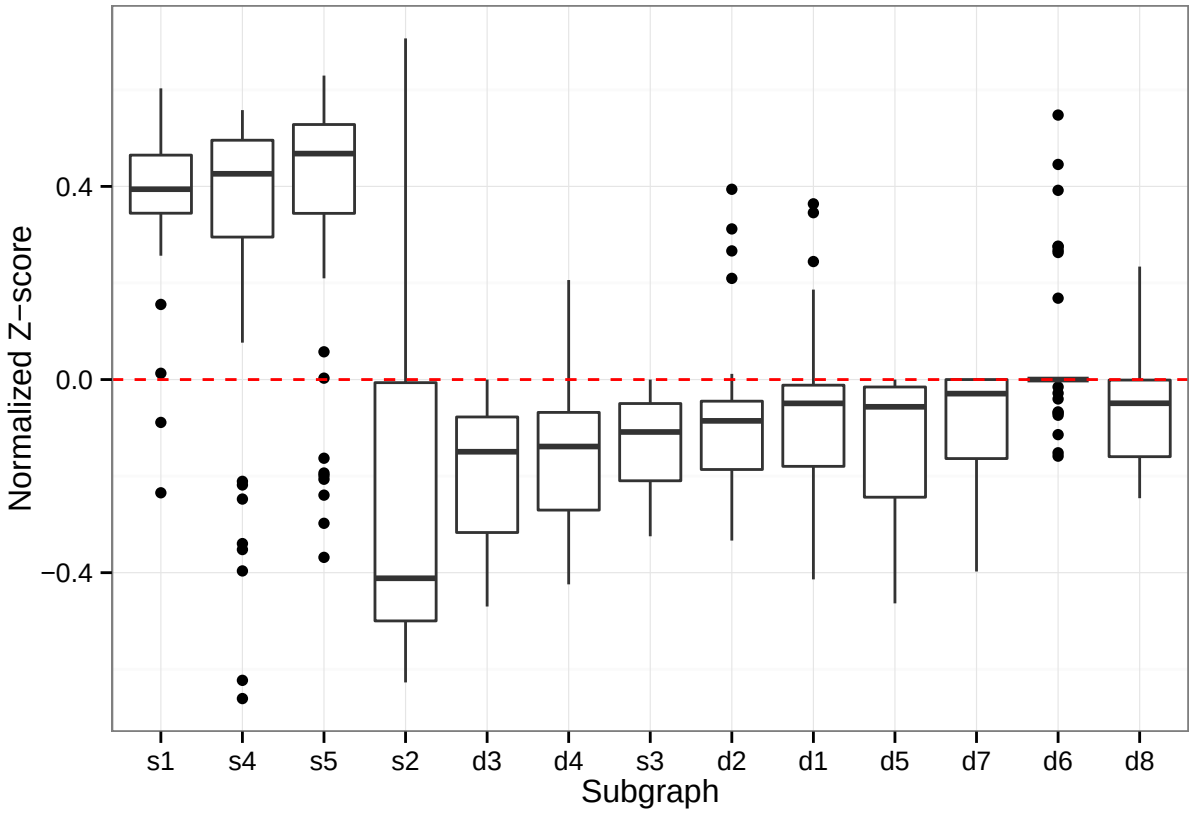
```
    Pearson's product-moment correlation

data:  sorted and med.z
t = 3.527, df = 11, p-value = 0.004737
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.2966 0.9130
sample estimates:
   cor
0.7285
```

Code for the figures

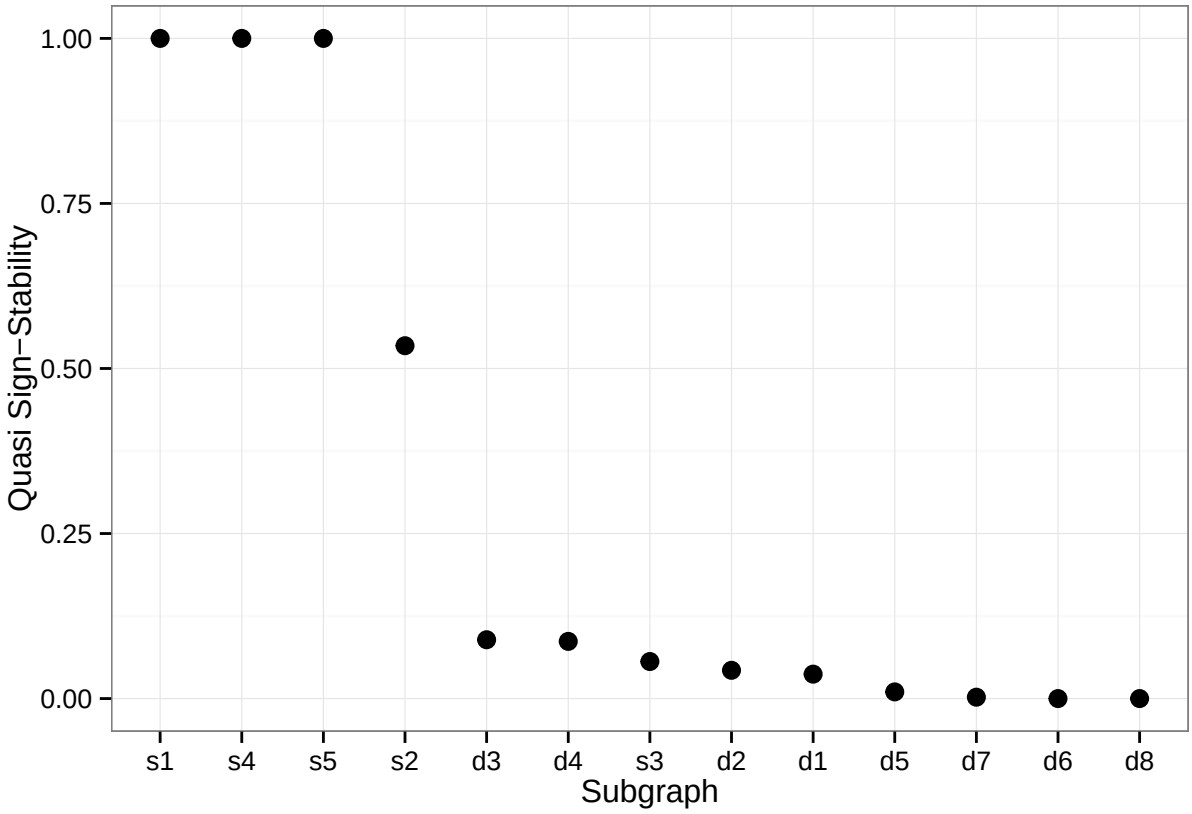
```
plot.df <- melt(profile[,names(sorted)])

fplot <- ggplot(plot.df, aes(x = Var2, y = value)) + geom_boxplot()
fplot <- fplot + geom_hline(aes(yintercept = 0), lty = 2, col = "red")
fplot <- fplot + theme_bw()
fplot + xlab("Subgraph") + ylab("Normalized Z-score")
```




```
sort.df <- melt(sorted)

qssplot <- ggplot(sort.df, aes(x = 1:13, y = value)) + geom_point(shape = 19, size = 3) + theme_bw()
qssplot + xlab("Subgraph") + ylab("Quasi Sign-Stability") + scale_x_discrete(limits=names(sorted))
```


1.00
0.75
0.50
0.25
0.00
Quasi Sign−Stability
s1
s4
s5
s2
d3
d4
s3
d2
d1
d5
d7
d6
d8
Subgraph