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

Jonathan J. Borrelli

Wednesday, July 23, 2014

Contents

1 Definitions **1**

1.1 Subgraph Library 1

1.2 Define required functions 2

2 Analysis **5**

2.1 Determining motif frequency 5

2.2 Determining Quasi Sign-Stability 8

3 Code for the figures **11**

3.1 Figure 1 11

3.2 Figure 2 12

3.3 Figure 3 12

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

1 Definitions

1.1 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")
```

1.2 Define required functions

1.2.1 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, d4 = triad.df$d4, d5 = triad.df$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 Curveball algorithm is available as supplemental information as part of the original publication. *Note if you want to use this code please cite the paper in which it was introduced:*

> Strona, G. et al. 2014. A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals. -Nat. Comm. 5: 4114. doi: 10.1038/ncomms5114

Their function takes a matrix and makes a swap (process descirbed in their paper) and returns the new matrix.

```
curve_ball <- function(m) {
    RC = dim(m)
    R = RC[1]
    C = RC[2]
    hp = list()
    for (row in 1:dim(m)[1]) {
        hp[[row]] = (which(m[row, ] == 1))
    }
    l_hp = length(hp)
    for (rep in 1:5 * l_hp) {
        AB = sample(1:l_hp, 2)
        a = hp[[AB[1]]]
        b = hp[[AB[2]]]
        ab = intersect(a, b)
        l_ab = length(ab)
        l_a = length(a)
        l_b = length(b)
        if ((l_ab %in% c(l_a, l_b)) == F) {
            tot = setdiff(c(a, b), ab)
            l_tot = length(tot)
            tot = sample(tot, l_tot, replace = FALSE, prob = NULL)
            L = l_a - l_ab
            hp[[AB[1]]] = c(ab, tot[1:L])
            hp[[AB[2]]] = c(ab, tot[(L + 1):l_tot])
        }

    }
    rm = matrix(0, R, C)
    for (row in 1:R) {
        rm[row, hp[[row]]] = 1
    }
    rm
}
```

The `curving` function is used to iteratively apply the Curveball algorithm to a single matrix. This function takes an adjacency matrix and number of iterations as inputs and returns a dataframe of motif frequencies.

```
curving <- function(adjmat, n) {
    mot <- motif_counter(list(graph.adjacency(adjmat)))
    newmat <- adjmat

    for (i in 1:n) {
        newmat <- curve_ball(newmat)
        m <- motif_counter(list(graph.adjacency(newmat)))
        mot <- rbind(mot, m)
    }
    return(mot[-1, ])
}
```

I added the additional constraints of maintaining the number of single, double, and self links in each matrix to the original curveball algorithm in the function `dblcan.curve`. This function takes a matrix and desired number of iterations as inputs and returns a dataframe of motif frequencies.

```
library(plyr)

nd <- function(gl) nrow(gl) - nrow(unique(aaply(gl, 1, sort)))
# determines the number of double links

dblcan.curve <- function(mat, iter) {
    mot <- motif_counter(list(graph.adjacency(mat)))

    el <- get.edgelist(graph.adjacency(mat))
    Ne <- nrow(el)
    dbl <- nd(el)
    can <- sum(diag(mat))

    for (i in 1:iter) {
        ed = TRUE
        dub = TRUE
        ca = TRUE
        while (ed || dub || ca) {
            mat2 <- curve_ball(mat)

            el2 <- get.edgelist(graph.adjacency(mat2))
            el3 <- unique(el2)
            Ne2 <- nrow(el3)
            dbl2 <- nd(el3)
            can2 <- sum(diag(mat2))

            ed <- Ne != Ne2
            dub <- dbl != dbl2
            ca <- can != can2

        }
        mat <- mat2

        mot <- rbind(mot, motif_counter(list(graph.adjacency(mat))))
    }
    return(M = mot[-1, ])
}
```

1.2.2 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)
}
```

2 Analysis

Load required packages

```
library(igraph)
library(ggplot2)
library(reshape2)
library(parallel)
library(doSNOW)
```

2.1 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`).

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

The first null model, the Curveball algorithm can be applied to all 50 webs in parallel. This code starts by registering a cluster utilizing 1 less than the number of cores on the computer (for this paper it was 7 cores). It then creates set of 30000 matrices and returns a list (length equal to the number of webs, 50 in this case) of dataframes of motif frequences.

```
cl <- makeCluster(detectCores() - 1)
clusterExport(cl, c("web.adj", "motif_counter", "curve_ball", "curving"))
registerDoSNOW(cl)
randos <- parLapply(cl, web.adj, curving, n = 30000)
stopCluster(cl)
```

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:

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

The normalized profile was then computed (as desribed in the methods):

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

```
means <- t(sapply(randos, colMeans))
stdevs <- t(sapply(randos, function(x) {
    apply(x, 2, sd)
}))

motfreq <- motif_counter(web.graphs)

zscore <- (motfreq - means)/stdevs

# Normalized z-scores
zscore.norm <- t(apply(zscore, 1, function(x) {
    x/sqrt(sum(x^2, na.rm = T))
}))
```

Below is the code used to run the Curveball algorithm with the additional constraints described above. *Note this can take a very long time to run*

```
# Use the filepath.sink variable to set the path for storing the sink
# information
filepath.sink <- my.sink.path
# Use the filepath.data variable to set the location for storing the
# dataframes of motif frequencies for each web
filepath.data <- my.data.path

cl <- makeCluster(detectCores() - 1)
clusterExport(cl, c("web.adj", "motif_counter", "curve_ball", "nd", "aaply",
    "filepath"))
registerDoSNOW(cl)
system.time(randos.t3 <- foreach(i = 1:length(web.adj)) %dopar% {
    sink(file = paste(filepath.sink, names(web.adj[i]), ".txt", collapse = ""))
    motfreq <- dblcan.curve(web.adj[[i]], iter = 30000)
    print(motfreq)
    sink()
    write.csv(motfreq, file = paste(filepath.data, names(web.adj[i]), ".csv",
        collapse = ""))
    return(motfreq)
})
stopCluster(cl)
```

The normalized z-score profile can be calculated the same as above:

```
means.t <- t(sapply(randos.t3, colMeans))
stdevs.t <- t(sapply(randos.t3, function(x) {
    apply(x, 2, sd)
}))

motfreq <- motif_counter(web.graphs)

zscore.t <- (motfreq - means.t)/stdevs.t

# Normalized z-scores
zscore.N <- t(apply(zscore.t, 1, function(x) {
    x/sqrt(sum(x^2, na.rm = T))
}))
```

Figure 1 is then a boxplot of the above normalized z-scores, reordered according to decreasing quasi sign-stability (see below). When using the modified Curveball algorithm, those webs that have no double links produce `NaN` when computing the z-score for those motifs. These are ignored in **Figure 1**, and the z-scores presented are with the `NaN`s removed.

2.2 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
```

2.2.1 Robustness to assumption of double link positives

I assumed that if there was a double link in the subgraph, that both the effect of the predator on the prey and the effect of the prey on the predator were positive, a (+/+) rather than a (+/-). Here I repeat the analysis described above, but instead assuming that the relative effects correspond to a (-/-).

```
mot.lst2 <- lapply(mot.lst, function(x) {
    for (i in 1:nrow(x)) {
        for (j in 1:ncol(x)) {
            if (x[i, j] == 1 && x[j, i] == 1) {
                x[i, j] <- x[i, j] * -1
                x[j, i] <- x[j, i] * -1
            } else {
                next
            }
        }
    }
    return(x)
})

set.seed(15)

n <- 10000
mot.stab2 <- eig.analysis(n, mot.lst2)
colnames(mot.stab2) <- names(mot.lst2)

mot.qss2 <- apply(mot.stab2, 2, function(x) {
    sum(x < 0)/n
})
sorted2 <- sort(mot.qss2, decreasing = T)
sorted2
```

```
    s1     s4     s5     d4     d3     s2     d2     d1     d7     d5
1.0000 1.0000 1.0000 0.8924 0.8914 0.5339 0.5042 0.5038 0.4890 0.3946
    d6     d8     s3
0.2440 0.2163 0.0558
```

2.2.2 Robustness to assumption of matrix fill distributions

I also tested how quasi sign-stability of the different subgraphs changed when varying the assumption of the relative impact of the predator on its prey and the prey on its predator.

I tested 7 additional distributions (all uniform) with different magnitudes.

```
params.u <- data.frame(pred1 = c(0, 0, 0, 0, 0, 0, 0, 0), pred2 = c(10, 10,
    10, 10, 10, 5, 3, 1), prey1 = c(-10, -5, -1, -0.1, -0.01, -1, -1, -1), prey2 = c(0,
    0, 0, 0, 0, 0, 0, 0))
parvals <- factor(paste(params.u[, 2], params.u[, 3], sep = "/"), levels = c("1/-1",
    "3/-1", "5/-1", "10/-1", "10/-0.01", "10/-0.1", "10/-1", "10/-5", "10/-10"))
```

Below is a function to take in the different parameters for the distributions and fill a matrix accordingly.

```
eigen_unif <- function(m, iter, params, self = -1) {
    # For when I want to use uniform distribution Params is dataframe of min and
    # max for relative impact of prey on pred and pred on prey
    ev <- c()
    for (i in 1:iter) {
        m1 <- apply(m, c(1, 2), function(x) {
            if (x == 1) {
                runif(1, params$pred1, params$pred2)
            } else if (x == -1) {
                runif(1, params$prey1, params$prey2)
            } else {
                0
            }
        })
        diag(m1) <- self
        ev[i] <- max(Re(eigen(m1)$values))
    }
    return(ev)
}
```

The following code loops through the dataframe of distributions and computes quasi sign-stability

```
test <- matrix(nrow = nrow(params.u), ncol = length(mot.lst))
for (i in 1:nrow(params.u)) {
    eigen.test <- lapply(mot.lst, eigen_unif, iter = 10000, params = params.u[i,
        ], self = runif(3, -1, 0))
    qss.test <- lapply(eigen.test, function(x) {
        sum(x < 0)/length(x)
    })
    test[i, ] <- unlist(qss.test)
}
```

2.2.3 Figure A1

```
colnames(test) <- c("s1", "s2", "s3", "s4", "s5", "d1", "d2", "d3", "d4", "d5",
    "d6", "d7", "d8")
test <- data.frame(test, parvals)
```

```
dat1 <- melt(test[, c(names(sorted), "parvals")])

ggplot(dat1, aes(x = variable, y = value)) + geom_point() + facet_wrap(~parvals,
    ncol = 4) + theme_bw() + xlab("Subgraph") + ylab("Quasi Sign-Stability")
```

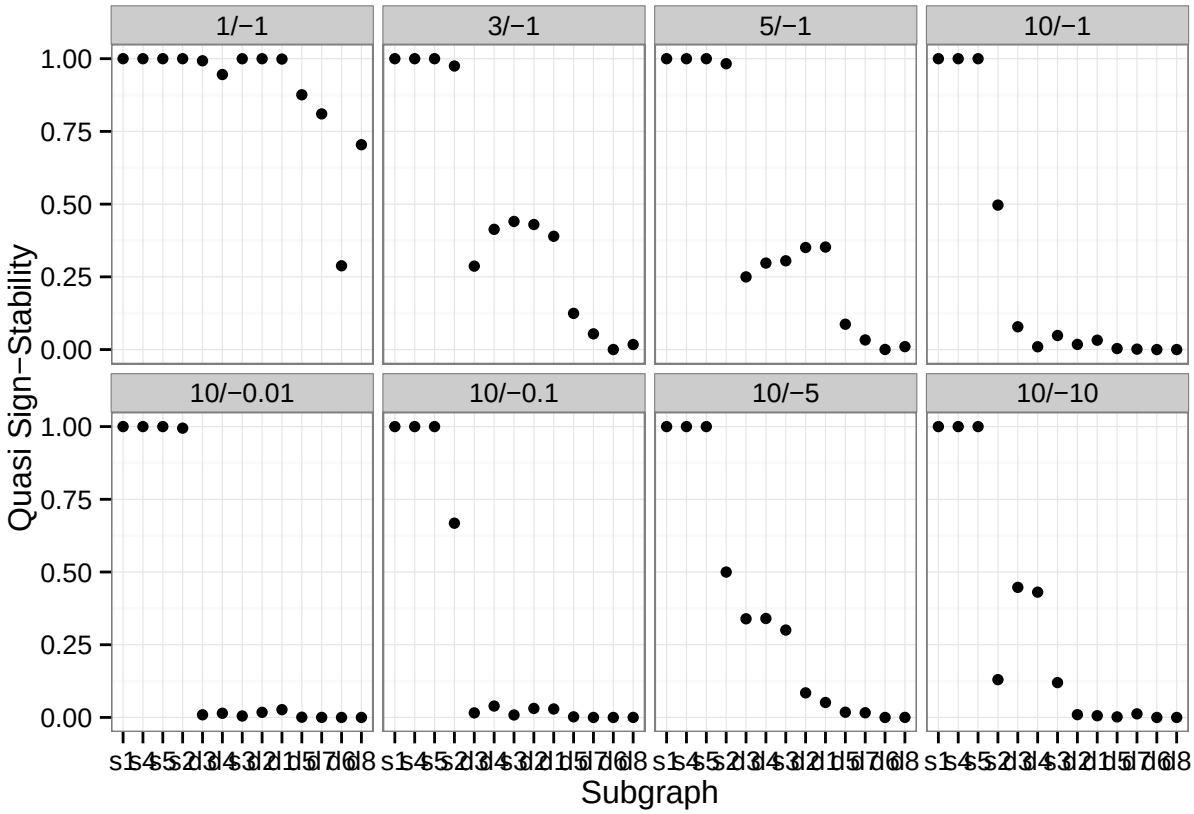
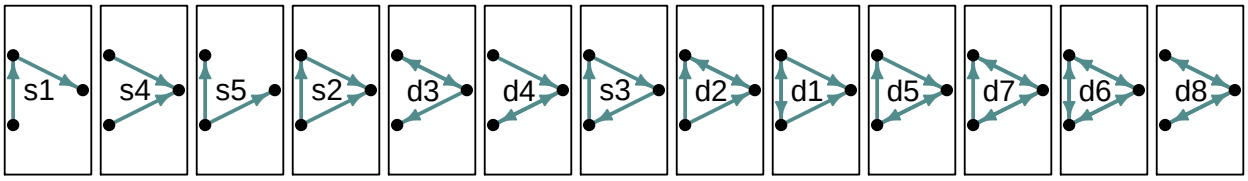


Figure A1: Robustness of quasi sign-stability to different distributions used to fill the random Jacobian matrix. Facet labels represent the maximum impact of the prey on the predator population (positive number) and maximum impact of the predator on the prey population (negative number)

3 Code for the figures

3.1 Figure 1

```
par(mfrow = c(1, 13), mar = c(0.2, 0.2, 0.2, 0.2))
for (i in 1:13) {
    plot.igraph(graph.adjacency(mot.lst[[which(names(mot.lst) == names(sorted[i]))]]),
        layout = layout.circle, edge.arrow.size = 0.5, vertex.size = 30, vertex.color = "black",
        vertex.label = NA, frame = T, edge.width = 2, edge.color = "darkslategray4")
    text(-0.25, 0, names(sorted[i]), cex = 1.5)
}
```

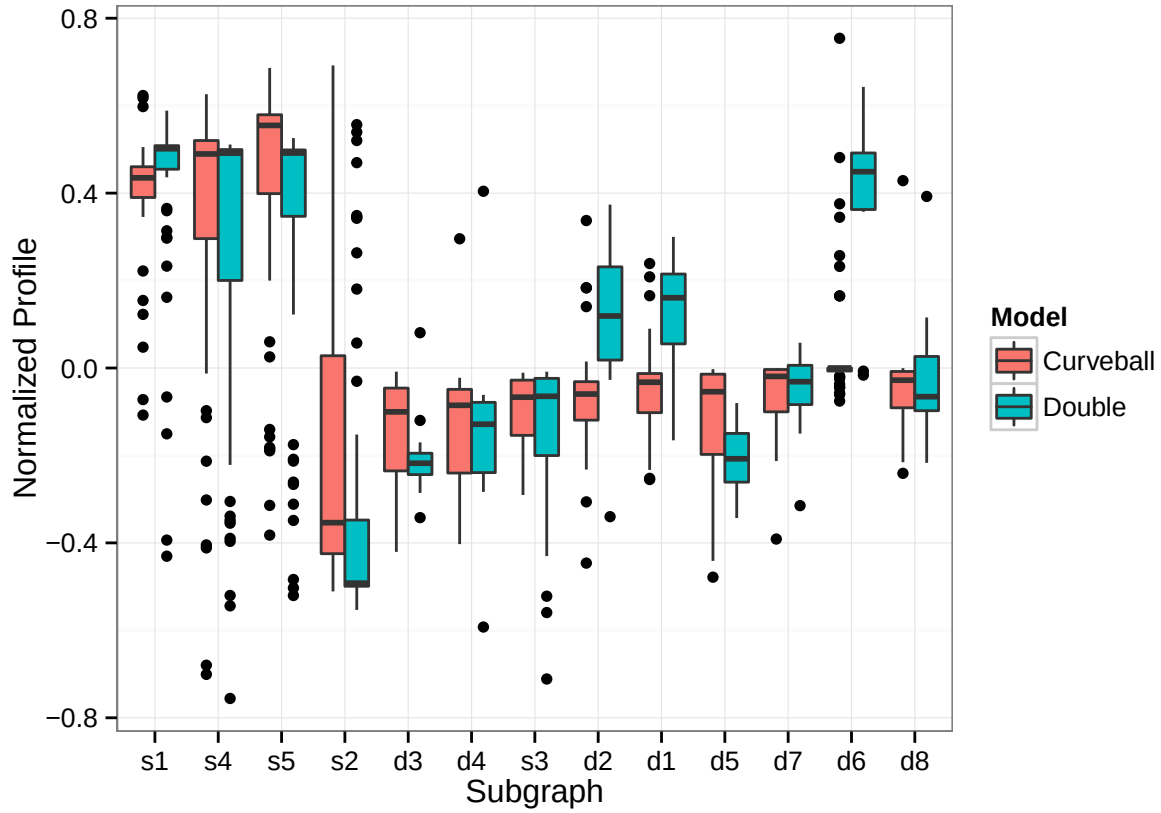


3.2 Figure 2

```
z1 <- cbind(Model = factor("Curveball"), melt(zscore.norm[, names(sorted)]))
z2 <- cbind(Model = factor("Double"), melt(zscore.N[, names(sorted)]))

z.both <- rbind(z1, z2)
ggplot(z.both, aes(x = Var2, y = value, fill = Model)) + geom_boxplot() + xlab("Subgraph") +
    ylab("Normalized Profile") + theme_bw()
```

```
Warning: Removed 308 rows containing non-finite values (stat_boxplot).
```



3.3 Figure 3

```
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))
```