Appendix S1: A guide to choosing and implementing reference models for social network analysis

Matthew Silk

02/02/2021

Please direct any questions about the examples presented in this R script to Matthew Silk (matthewsilk@outlook.com)

Section 1 – Preparation

First we are going to prepare the R environment and load the necessary packages for our case study. If you want to explore the variability in the networks and result possible you can change the number in the set.seed() function to produce different networks.

Note that throughout this script we use an edited version of the asnipe get_network2 function that doesn't print messages

```
#Set seed for reproducibility
#Can be changed to produce different networks and explore variability in results
set.seed(35)

##load packages
library(asnipe)
library(igraph)
library(boot)
library(prodlim)
library(sna)
library(assortnet)
library(blockmodels)
library(ergm)
library(ergm.count)
library(tnet)
library(vegan)
```

Section 2 – Network Generation

Creating a population of burbils with social networks

Burbils live in open habitats throughout the world. They form fission-fusion societies characterised by stable social groups that roost together but fission into smaller subgroups when foraging during the day. Foraging subgroups from different groups occasionally meet and intermingle creating opportunities for between-group interactions. These between-group associations are more likely if the two Burbil groups belong to the same "clan". Burbil groups vary in size and we are unsure whether groups of different sizes have similar social network structures. Groups also contain two unique colour morphs: burbils with red noses, and those with orange noses. As well as being able to identify individual burbils (which we use to construct their social

networks!), we are also able to distinguish male and female burbils as well as those from three distinct age classes (adults, subadults and juveniles). We know that burbils are involved in both dominance interactions and affiliative interactions with group-mates. We suspect they may have a dominance hierarchy, but we don't know this for sure. We have a lot to find out!

A burbil

Section 2.1 – Generate population network

In this section of the code we create our burbil society (starting with the association network), explaining what we do as we go along. With practice it should be possible to change some of the numbers in this code to change the nature of social relationships in your burbil society.

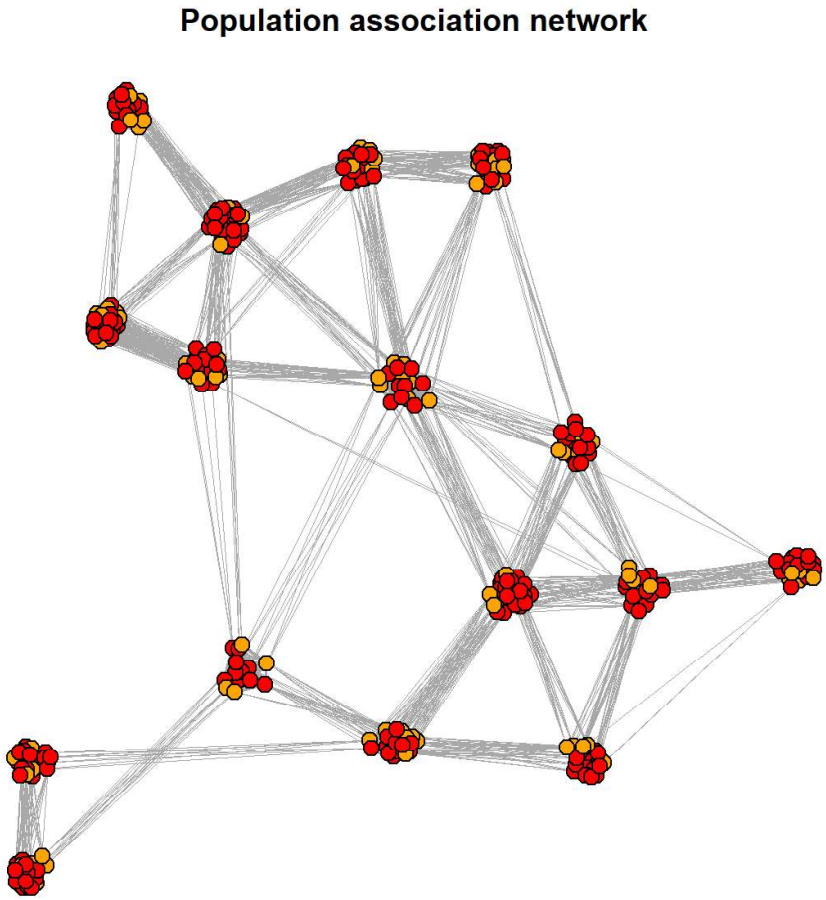
```
#Set the mean group size
GS<-20

#Here we create a grid of locations for our observations
x<-seq(3,18,1)
y<-seq(3,18,1)
locs<-expand.grid(x,y)
names(locs)<-c("x","y")

#Here we assign coordinates to our groups. We create 9 groups in total.
group_locs<-locs[locs$x%%4==0&locs$y%%4==0,]

#Here we store the total number of groups
n_groups<-dim(group_locs)[1]

#Here we create three distinct clans of burbils. This will effect associations bet
ween members of different groups
group_clans<-sample(c("A","B","C"),n_groups,replace=TRUE)

#Set the probability of burbils from the same clan intermingling if they happen to
forage at the same location
p_wc<-1
#Set the probability of burbils from different clans intermingling if they happen
 to forage at the same location
```

```r
p_bc<-0.4

#Create a list to store individual IDs
indss<-list()

#Create a list to store group sizes
gss<-list()

#Create a list to store the sex of each individual
sexes<-list()

#Create a list to store the age of each individual
ages<-list()

#Create a list to store the nose colour of each individual
noses<-list()

#Create a list to store information on which day a subgroup is observed on
daysl<-list()

#Create a list to store a group-by-individual matrix for each burbil group
gbis<-list()

#Set the mean number of subgroups observed for each group each day
sg_mn<-5

#Set the strength of assortativity based on nose colour
#Set a number between 0 and 1
sg_ass<-0.15

#Generate association data within each burbil group
for(j in 1:n_groups){

#individual identities
inds<-seq(1,rpois(1,GS),1)
indss[[j]]<-inds

#group size
gs<-length(inds)
gss[[j]]<-gs

#sex
sex<-sample(c("M","F"),gs,replace=TRUE)
sexes[[j]]<-sex

#age
age<-sample(c("AD","SUB","JUV"),gs,replace=TRUE,prob=c(0.6,0.2,0.2))
ages[[j]]<-age

#nose
nose<-sample(c("RED","ORANGE"),gs,replace=TRUE,prob=c(0.7,0.3))
noses[[j]]<-nose

#---------------------------------

#Define number of subgroups on the first day
n_sg<-rpois(1,sg_mn-1)+1
```

```
#find halfway point
max_red<-floor(n_sg/2)

#Sample subgroups on the first day
subgroups1<-sample(n_sg,sum(nose=="RED"),replace=TRUE,prob=c(rep(0.5+sg_ass,max_re
d),rep(0.5-sg_ass,n_sg-max_red)))
subgroups2<-sample(n_sg,sum(nose=="ORANGE"),replace=TRUE,prob=c(rep(0.5-sg_ass,max
_red),rep(0.5+sg_ass,n_sg-max_red)))

subgroups<-rep(NA,gs)
subgroups[nose=="RED"]<-subgroups1
subgroups[nose=="ORANGE"]<-subgroups2

#Store relevant information in the group-by-individual matrix and days vector
gbi<-matrix(0,nc=gs,nr=n_sg)
gbi[cbind(subgroups,seq(1,gs,1))]<-1
days<-rep(1,nrow(gbi))

#Repeat process over 100 days of observations
for(i in 2:100){

  n_sg<-rpois(1,sg_mn-1)+1

  #find halfway point
  max_red<-floor(n_sg/2)

  subgroups1<-sample(n_sg,sum(nose=="RED"),replace=TRUE,prob=c(rep(0.5+sg_ass,max_
red),rep(0.5-sg_ass,n_sg-max_red)))
  subgroups2<-sample(n_sg,sum(nose=="ORANGE"),replace=TRUE,prob=c(rep(0.5-sg_ass,m
ax_red),rep(0.5+sg_ass,n_sg-max_red)))

  subgroups<-rep(NA,gss[[j]])
  subgroups[nose=="RED"]<-subgroups1
  subgroups[nose=="ORANGE"]<-subgroups2

  tgbi<-matrix(0,nc=gs,nr=n_sg)
  tgbi[cbind(subgroups,seq(1,gs,1))]<-1
  days<-c(days,rep(i,nrow(tgbi)))
  gbi<-rbind(gbi,tgbi)
}

#We edit the group-by-individual matrix and days vector to delete any "empty" grou
ps
gbi2<-gbi[rowSums(gbi)>0,]
days<-days[rowSums(gbi)>0]
gbi<-gbi2

#We could create and plot the network for each burbil group
#(NOT RUN HERE)
#net<-get_network2(gbi)
#net2<-graph.adjacency(net,mode="undirected",weighted=TRUE)
#plot(net2,vertex.color=noses[[j]],edge.width=(edge_attr(net2)$weight*10)^2)

daysl[[j]]<-days
gbis[[j]]<-gbi
```

```
}

#We now go through and assign a location to every subgroup
sglocs<-list()
for(i in 1:n_groups){
  tx<-rep(NA,dim(gbis[[i]])[1])
  ty<-rep(NA,dim(gbis[[i]])[1])
  sglocs[[i]]<-data.frame(tx,ty)
  names(sglocs[[i]])<-c("x","y")
  sglocs[[i]]$x<-group_locs[i,1]+round(rnorm(dim(gbis[[i]])[1],0,2))
  sglocs[[i]]$y<-group_locs[i,2]+round(rnorm(dim(gbis[[i]])[1],0,2))
}

#Vector recording number of individuals in each group
n_inds<-numeric()
for(i in 1:n_groups){
  n_inds[i]<-dim(gbis[[i]])[2]
}

#Calculate total individuals in the population
n_tot<-sum(n_inds)

#Population-level individual identities
inds_tot<-seq(1,n_tot,1)

#Information on each individual's group membership
g_tot<-rep(seq(1,n_groups,1),n_inds)

#Information on each individual's within-group identity
gi_tot<-seq(1,n_inds[1],1)
for(i in 2:n_groups){
  gi_tot<-c(gi_tot,seq(1,n_inds[i],1))
}

#We now calculate the full population association network
full_net<-matrix(0,nr=n_tot,nc=n_tot)

#Counts up between-group associations
for(i in 1:100){
  for(j in 1:(n_groups-1)){
    for(k in (j+1):n_groups){
      tA<-paste0(sglocs[[j]][,1],"-",sglocs[[j]][,2])
      tB<-paste0(sglocs[[k]][,1],"-",sglocs[[k]][,2])
      tA2<-tA[daysl[[j]]==i]
      tB2<-tB[daysl[[k]]==i]
      tt<-match(tA2,tB2)
      if(sum(is.na(tt))<length(tt)){
      if(group_clans[j]==group_clans[k]){same<-rbinom(1,1,p_wc)}
      if(group_clans[j]!=group_clans[k]){same<-rbinom(1,1,p_bc)}
      if(same==1){
        paste(i,j,k)
        for(m in length(tt)){
          if(is.na(tt[m])==FALSE){
            tsg1<-which(tA==tA2[m]&daysl[[j]]==i)
            tsg2<-which(tB==tB2[tt[m]]&daysl[[k]]==i)
            tid1<-which(gbis[[j]][tsg1,]==1)
            tid2<-which(gbis[[k]][tsg2,]==1)
```

```
            tid1a<-inds_tot[g_tot==j&gi_tot%in%tid1]
            tid2a<-inds_tot[g_tot==k&gi_tot%in%tid2]
            full_net[tid1a,tid2a]<-full_net[tid1a,tid2a]+1
            full_net[tid2a,tid1a]<-full_net[tid1a,tid2a]
          }
        }
      }
      }
    }
  }
}

#converts between group associations to simple ratio indicess (SRIs)
for(i in 1:(nrow(full_net)-1)){
  for(j in (i+1):nrow(full_net)){
    full_net[i,j]<-full_net[i,j]/(200-full_net[i,j])
    full_net[j,i]<-full_net[i,j]
  }
}

#Adds within-group associations to the population network
for(i in 1:n_groups){
  full_net[inds_tot[g_tot==i],inds_tot[g_tot==i]]<-get_network2(gbis[[i]])
}

#Plots the population social network
full_net2<-graph.adjacency(full_net,mode="undirected",weighted=TRUE)
par(mar=c(0,0,2,0))
plot(full_net2,vertex.color=unlist(noses),vertex.label=NA,vertex.size=4,edge.width
=(edge_attr(full_net2)$weight*10)^2,main="Population association network")
```




```
par(mar=c(5,6,2,2))
```

Section 2.2 – Generate within-group networks of behavioural interactions

We now focus in on a single burbil group (group 1) and generate data on dominance interactions and affiliative behaviours. We are going to generate different structure in these interactions that we will attempt to uncover in subsequent analyses.

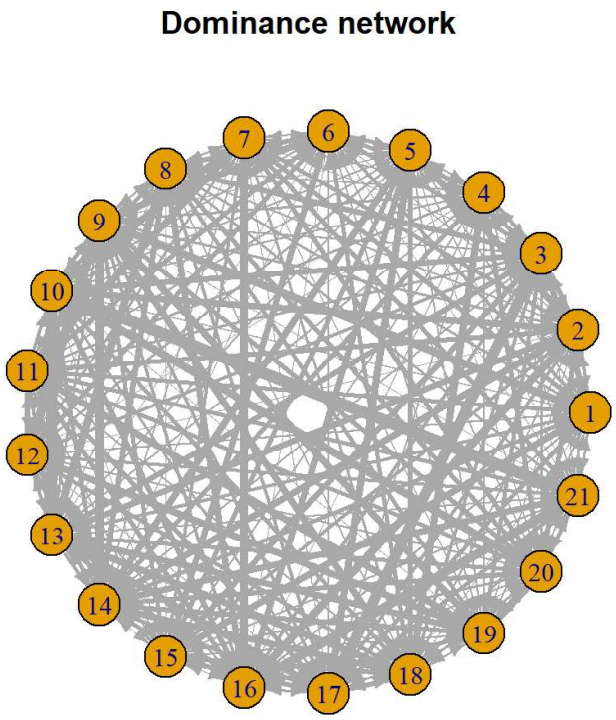
Another important feature is that dominance and affiliative interactions are only possible within subgroups. We use scan sampling of subgroups (as they are small) to record all interactions occurring. The number of interactions recorded depends on the size of the subgroup. As diligent researchers, we record the day and subgroup of all interactions.

Create dominance interactions

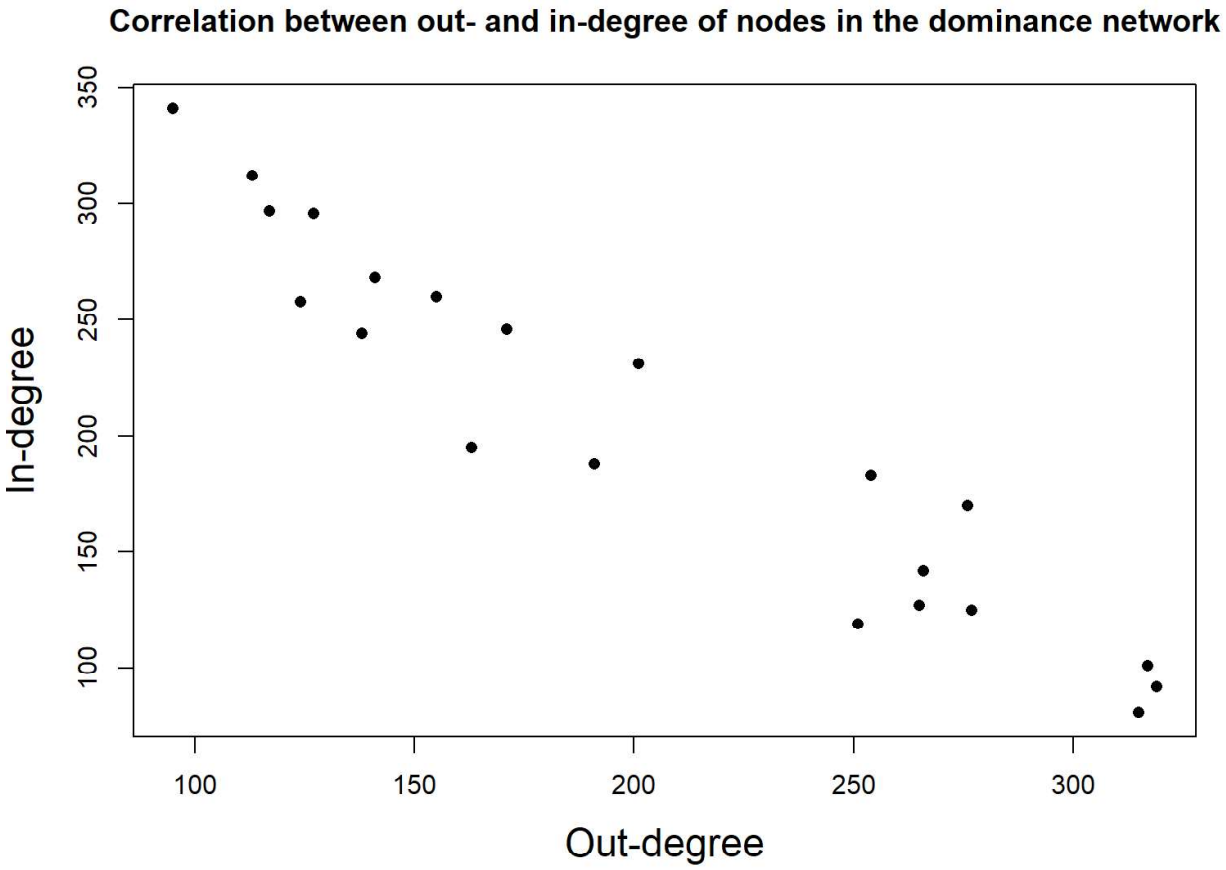
```
#individual identities
inds1<-indss[[1]]

#group size
gs1<-gss[[1]]
```

```
#sex
sex1<-sexes[[1]]

#age
age1<-ages[[1]]

#nose
nose1<-noses[[1]]

gbi<-gbis[[1]]

#Set-up vectors to store results
GROUP<-numeric()
WINNER<-numeric()
LOSER<-numeric()

#Define the resource holding potential of different individuals
RHP_ad<-1
RHP_sub<-0
RHP_juv<- -1
RHP_M<--0.5
RHP_resid<-0.2
RHPs1<-rnorm(gs1,RHP_ad*(age1=="AD")+RHP_sub*(age1=="SUB")+RHP_juv*(age1=="JUV")+R
HP_M*(sex1=="M"),RHP_resid)

#Define the mean number of interactions observed per individual in a subgroup
m_nipi<-2

#record which group the interactions occur in
grD<-numeric()

#Generate dominance interaction data
c<-1
for(g in 1:nrow(gbi)){
if(rowSums(gbi)[g]>1){
nipi<-rpois(1,m_nipi)
indivs<-which(gbi[g,]==1)
ni<-nipi*length(indivs)
for(n in 1:ni){
i1<-sample(indivs,1)
ifelse(rowSums(gbi)[g]==2,i2<-indivs[indivs!=i1],i2<-sample(indivs[indivs!=i1],1))
winner<-rbinom(1,1,inv.logit(RHPs1[i1]-RHPs1[i2]))
GROUP[c]<-g
if(winner==1){
  WINNER[c]<-i1
  LOSER[c]<-i2
}
if(winner==0){
  WINNER[c]<-i2
  LOSER[c]<-i1
}
grD[c]<-g
c<-c+1
}
}
}
```

```
#Create the dominance network in igraph format
dom_net<-graph_from_edgelist(cbind(WINNER,LOSER), directed = TRUE)
E(dom_net)$weight <- 1
dom_net<-simplify(dom_net, edge.attr.comb=list(weight="sum"))

#Plot the dominance network that results (it is densely connected and so the netwo
rk plot isn't especially informative)
plot(dom_net,edge.width=log(edge_attr(dom_net)$weight,10)^5,layout=layout_in_circl
e,main="Dominance network",edge.arrow.size=0.5)
```



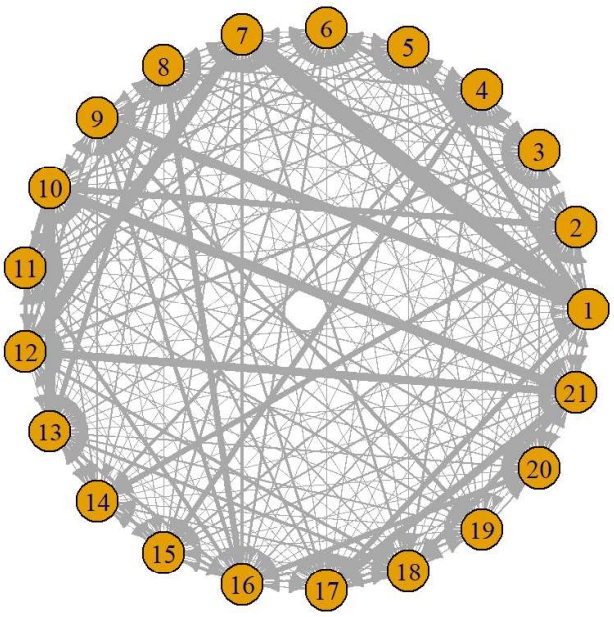

```
#To show that our code to generate the dominance network works we plot the relatio
nship between in-strength and out-strength and it is negatively correlated as woul
d be expected for a linear dominance hierarchy
plot(strength(dom_net,mode="out"),strength(dom_net,mode="in"),pch=16,xlab="Out-deg
ree",ylab="In-degree",cex.lab=1.5,cex.axis=1,main="Correlation between out- and in
-degree of nodes in the dominance network")
```



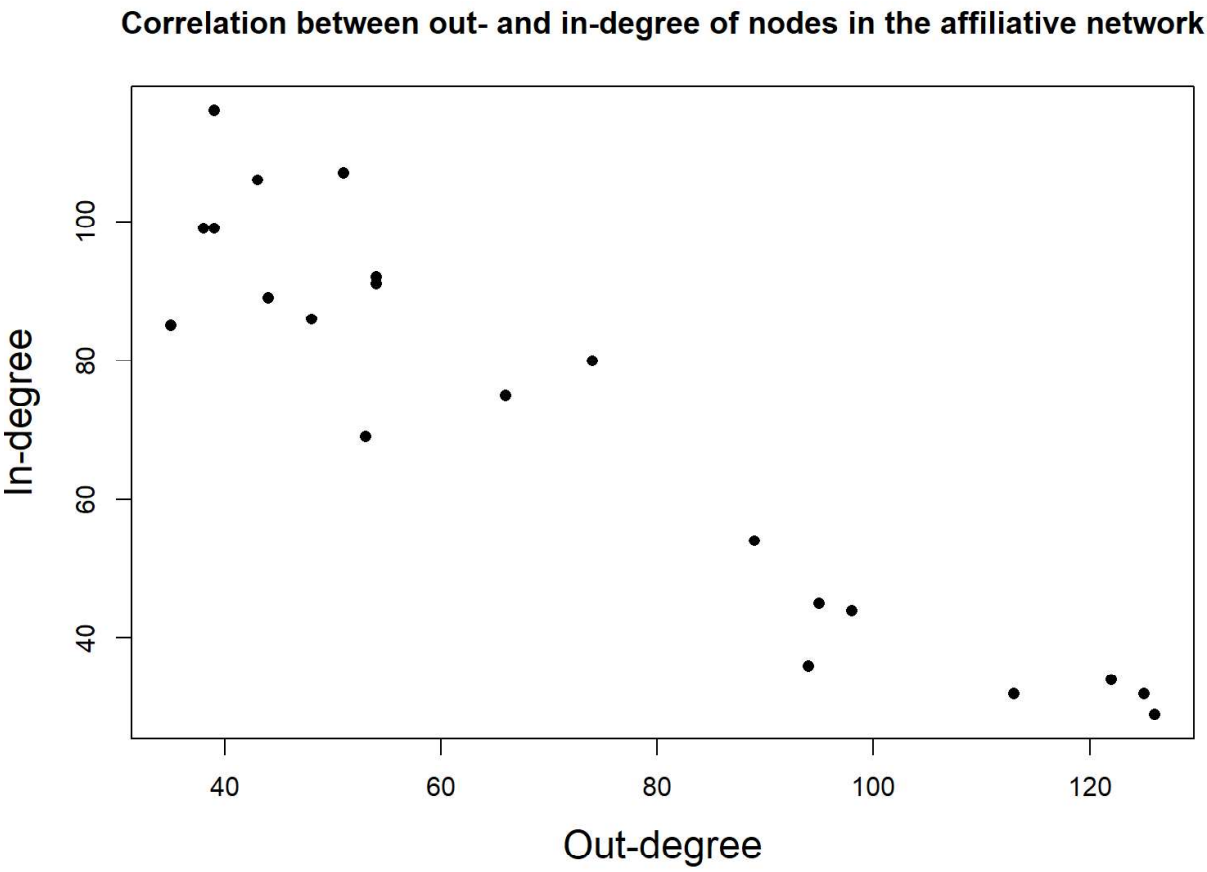
Create affiliative interactions

```
#We use an equivalent approach as for dominance networks so have kept much of the
 coding the same (hence the mismatch in names)

#Set-up vectors to store results
GROUP<-numeric()
GIV<-numeric()
REC<-numeric()

#Define the tendency of different individuals to initiate affiliative interactions
AHP_ad<- -1
AHP_sub<- -1
AHP_juv<-1
AHP_M<-0
AHP_nose<-1
AHP_resid<-0.2
AHPs1<-rnorm(gs1,AHP_ad*(age1=="AD")+AHP_sub*(age1=="SUB")+AHP_juv*(age1=="JUV")+A
HP_M*(sex1=="M"),AHP_resid)

#Define the mean number of interactions observed per individual in a subgroup
m_nipi<-0.5

#record which group interactions occur in
grA<-numeric()
```

```
#Generate affiliative interaction data
c<-1
for(g in 1:nrow(gbi)){
  if(rowSums(gbi)[g]>1){
    nipi<-rpois(1,m_nipi)
    indivs<-which(gbi[g,]==1)
    ni<-nipi*length(indivs)
    for(n in 1:ni){
      i1<-sample(indivs,1)
      ifelse(rowSums(gbi)[g]==2,i2<-indivs[indivs!=i1],i2<-sample(indivs[indivs!=i
1],1))
      tn<-0
      if(nose1[i1]==nose1[i2]){tn<-1}
      winner<-rbinom(1,1,inv.logit(AHPs1[i1]-AHPs1[i2]+tn))
      GROUP[c]<-g
      if(winner==1){
        GIV[c]<-i1
        REC[c]<-i2
      }
      if(winner==0){
        GIV[c]<-i2
        REC[c]<-i1
      }
      grA[c]<-g
      c<-c+1
    }
  }
}

#Create the affiliative network in igraph format
aff_net<-graph_from_edgelist(cbind(GIV,REC), directed = TRUE)
E(aff_net)$weight <- 1
aff_net<-simplify(aff_net, edge.attr.comb=list(weight="sum"))

#Plot the affiliative network that results
plot(aff_net,edge.width=log(edge_attr(aff_net)$weight,6)^5,layout=layout_in_circl
e,main="Affiliative network",edge.arrow.size=0.5)
```



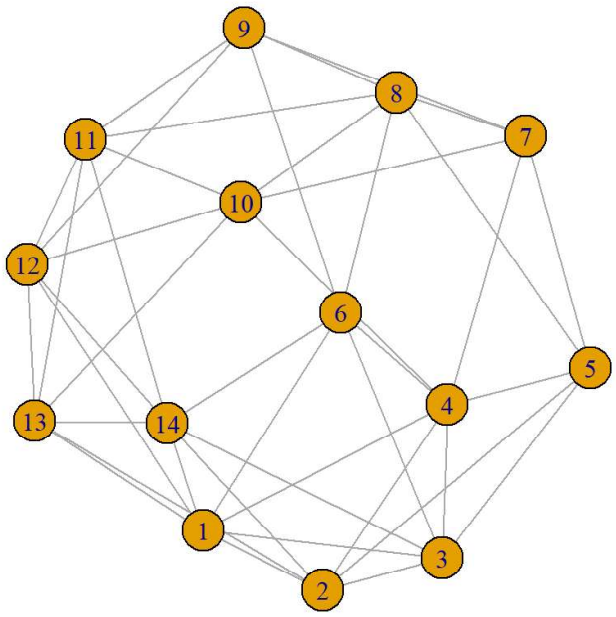

```
#Plot the same correlation used for dominance networks
plot(strength(aff_net,mode="out"),strength(aff_net,mode="in"),pch=16,xlab="Out-degree",ylab="In-degree",cex.lab=1.5,cex.axis=1,main="Correlation between out- and in-degree of nodes in the affiliative network")
```



Section 2.3 – Generate huddling networks

Data were also collected on the huddling networks of two burbil groups while they were roosting during summer and winter. These data can be used to test if the huddling networks differ between small and large groups. We simulate these data here.

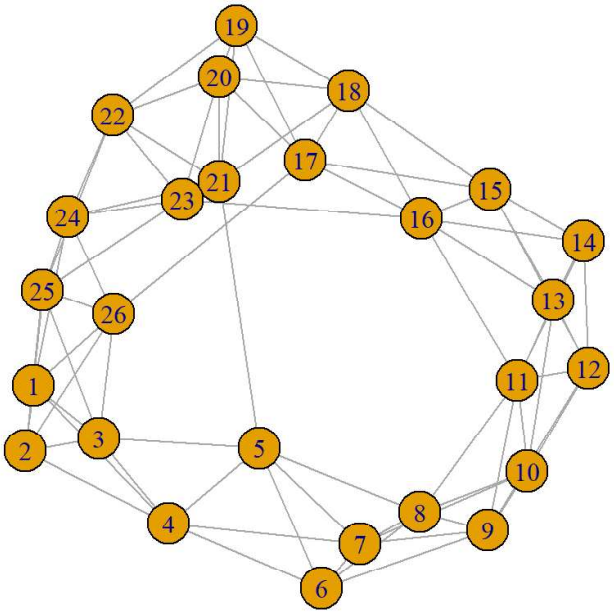
```
sm_g<-which.min(n_inds)
bi_g<-which.max(n_inds)

#Generate "roosting/huddling network of burbils in the smallest group in the summe
r
hud_netSM<-sample_smallworld(dim=1, size=gss[[sm_g]], nei=3, p=0.05, loops = FALSE
, multiple = FALSE)

#Plot network
plot(hud_netSM,main="Huddling network in small group")
```




```
#Calculate betweenness of network
igraph::betweenness(hud_netSM)
```

```
##  [1] 3.821429 3.142857 2.059524 6.535714 3.059524 5.380952 2.166667 4.392857
##  [9] 2.547619 5.214286 3.214286 2.976190 2.654762 4.833333
```

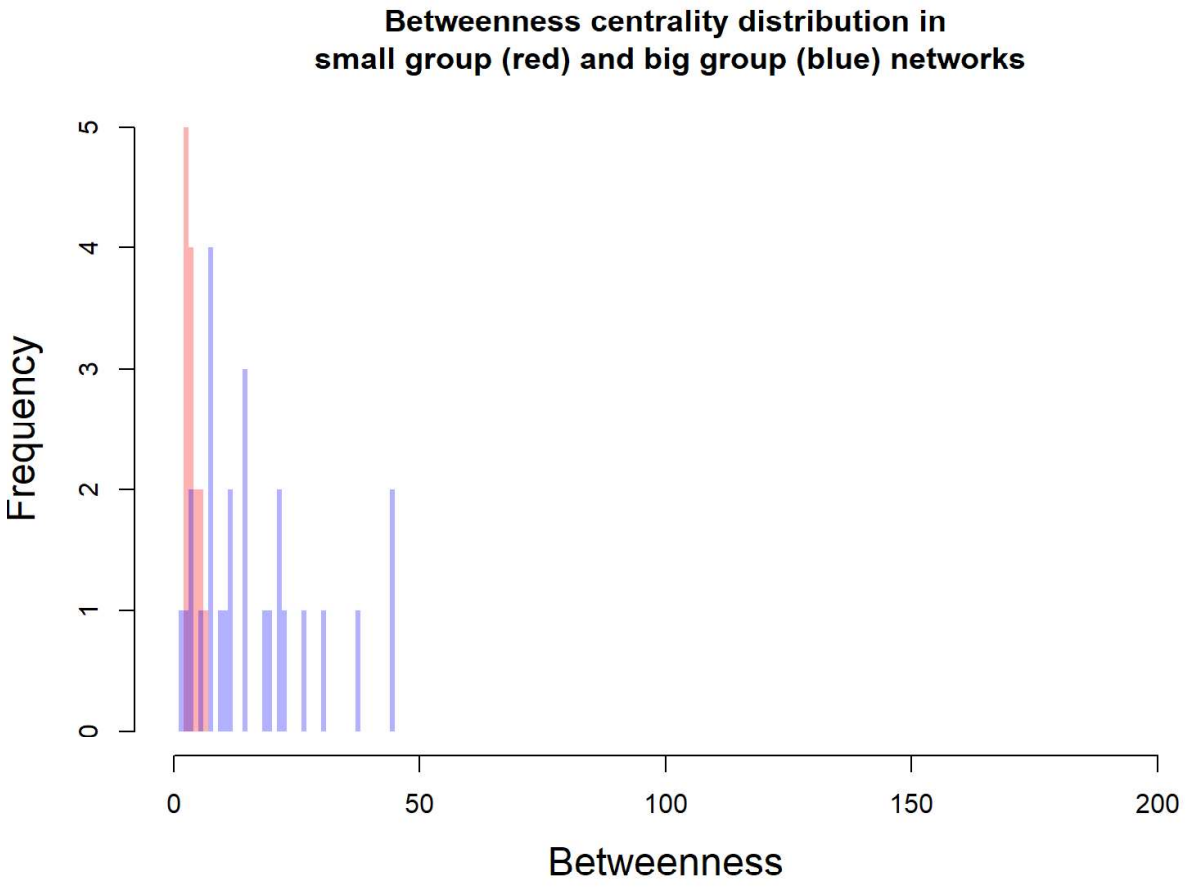
```
##----------------------------------------
##----------------------------------------

#Generate "roosting/huddling network of burbils in the biggest group  in the summe
r
hud_netBI<-sample_smallworld(dim=1, size=gss[[bi_g]], nei=3, p=0.05, loops = FALSE
, multiple = FALSE)

#Plot network
plot(hud_netBI,main="Huddling network in big group")
```




```
#Calculate betweenness of network
igraph::betweenness(hud_netBI)
```

```
##  [1]  7.262572  3.457215 11.813276 22.990693 44.068957  7.520382 18.858081
##  [8] 14.385155 11.266270 10.802778 21.436941  9.367496  7.158929  1.370635
## [15] 19.455988 44.348846 30.635191 14.511310  2.268651  3.560317 37.887302
## [22]  5.767857 26.572817  7.732738 14.036706 21.462897
```

```
#Examine differences in betweenness by inspecting histograms
hist(igraph::betweenness(hud_netSM),breaks=seq(0,200,1),col=rgb(1,0,0,0.3),border=
NA,xlab="Betweenness",cex.lab=1.5,cex.axis=1,main="Betweenness centrality distribu
tion in\n small group (red) and big group (blue) networks")
hist(igraph::betweenness(hud_netBI),breaks=seq(0,200,1),col=rgb(0,0,1,0.3),border=
NA,add=TRUE,cex.lab=1.5,cex.axis=1,main="")
```




```
##----------------------------------------
##----------------------------------------

#Generate "roosting/huddling network of burbils in the smallest group in the winte
r
hud_netSM_w<-erdos.renyi.game(n=gss[[sm_g]], p=0.3, loops = FALSE, multiple = FALS
E)
hud_netBI_w<-erdos.renyi.game(n=gss[[bi_g]], p=0.3, loops = FALSE, multiple = FALS
E)
```

Section 2.4 – A second population network

We have also been sent association data from a similar but smaller burbil population by a colleague. They want to know whether their burbil population has a similar network structure to ours.

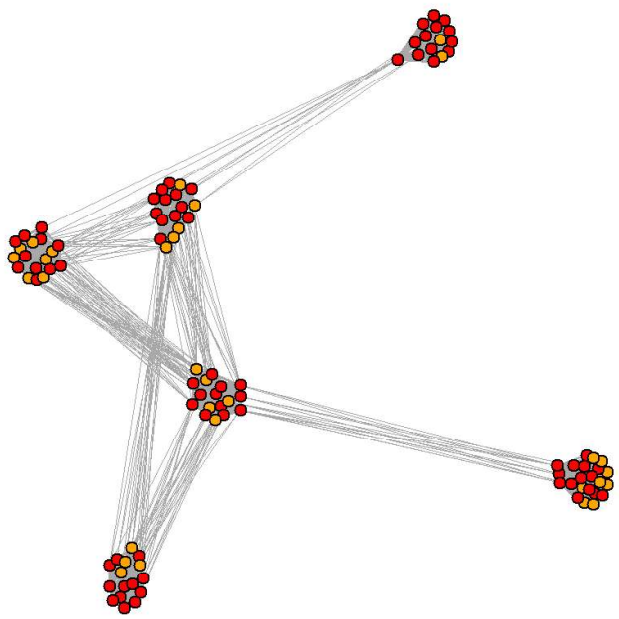
```
#Set the mean group size
GS_B<-20

#Here we create a grid of locations for our observations
x_B<-seq(3,13,1)
y_B<-seq(3,9,1)
locs_B<-expand.grid(x_B,y_B)
names(locs_B)<-c("x","y")
```

```r
#Here we assign coordinates to our groups. We create 9 groups in total.
group_locs_B<-locs_B[locs_B$x%%4==0&locs_B$y%%4==0,]

#Here we store the total number of groups
n_groups_B<-dim(group_locs_B)[1]

#Here we create three distinct clans of burbils. This will effect associations bet
ween members of different groups
group_clans_B<-sample(c("A","B","C"),n_groups_B,replace=TRUE)

#Set the probability of burbils from the same clan intermingling if they happen to
forage at the same location
p_wc_B<-1
#Set the probability of burbils from different clans intermingling if they happen
 to forage at the same location
p_bc_B<-0.4

#Create a list to store individual IDs
indss_B<-list()

#Create a list to store group sizes
gss_B<-list()

#Create a list to store the sex of each individual
sexes_B<-list()

#Create a list to store the age of each individual
ages_B<-list()

#Create a list to store the nose colour of each individual
noses_B<-list()

#Create a list to store information on which day a subgroup is observed on
daysl_B<-list()

#Create a list to store a group-by-individual matrix for each burbil group
gbis_B<-list()

#Set the mean number of subgroups observed for each group each day
sg_mn_B<-5

#Set the strength of assortativity based on nose colour
#Set a number between 0 and 1
sg_ass_B<-0.1

#Generate association data within each burbil group
for(j in 1:n_groups_B){

#individual identities
inds_B<-seq(1,rpois(1,GS_B),1)
indss_B[[j]]<-inds_B

#group size
gs_B<-length(inds_B)
gss_B[[j]]<-gs_B

#sex
```

```
sex_B<-sample(c("M","F"),gs_B,replace=TRUE)
sexes_B[[j]]<-sex_B

#age
age_B<-sample(c("AD","SUB","JUV"),gs_B,replace=TRUE,prob=c(0.6,0.2,0.2))
ages_B[[j]]<-age_B

#nose
nose_B<-sample(c("RED","ORANGE"),gs_B,replace=TRUE,prob=c(0.7,0.3))
noses_B[[j]]<-nose_B

#---------------------------------

#Define number of subgroups on the first day
n_sg_B<-rpois(1,sg_mn_B-1)+1

#find halfway point
max_red_B<-floor(n_sg_B/2)

#Sample subgroups on the first day
subgroups1_B<-sample(n_sg_B,sum(nose_B=="RED"),replace=TRUE,prob=c(rep(0.5+sg_ass_
B,max_red_B),rep(0.5-sg_ass_B,n_sg_B-max_red_B)))
subgroups2_B<-sample(n_sg_B,sum(nose_B=="ORANGE"),replace=TRUE,prob=c(rep(0.5-sg_a
ss_B,max_red_B),rep(0.5+sg_ass_B,n_sg_B-max_red_B)))

subgroups_B<-rep(NA,gs_B)
subgroups_B[nose_B=="RED"]<-subgroups1_B
subgroups_B[nose_B=="ORANGE"]<-subgroups2_B

#Store relevant information in the group-by-individual matrix and days vector
gbi_B<-matrix(0,nc=gs_B,nr=n_sg_B)
gbi_B[cbind(subgroups_B,seq(1,gs_B,1))]<-1
days_B<-rep(1,nrow(gbi_B))

#Repeat process over 100 days of observations
for(i in 2:100){

  n_sg_B<-rpois(1,sg_mn_B-1)+1

  #find halfway point
  max_red_B<-floor(n_sg_B/2)

  subgroups1_B<-sample(n_sg_B,sum(nose_B=="RED"),replace=TRUE,prob=c(rep(0.5+sg_as
s_B,max_red_B),rep(0.5-sg_ass_B,n_sg_B-max_red_B)))
  subgroups2_B<-sample(n_sg_B,sum(nose_B=="ORANGE"),replace=TRUE,prob=c(rep(0.5-sg
_ass_B,max_red_B),rep(0.5+sg_ass_B,n_sg_B-max_red_B)))

  subgroups_B<-rep(NA,gss_B[[j]])
  subgroups_B[nose_B=="RED"]<-subgroups1_B
  subgroups_B[nose_B=="ORANGE"]<-subgroups2_B

  tgbi_B<-matrix(0,nc=gs_B,nr=n_sg_B)
  tgbi_B[cbind(subgroups_B,seq(1,gs_B,1))]<-1
  days_B<-c(days_B,rep(i,nrow(tgbi_B)))
  gbi_B<-rbind(gbi_B,tgbi_B)
}
```

```
#We edit the group-by-individual matrix and days vector to delete any "empty" grou
ps
gbi2_B<-gbi_B[rowSums(gbi_B)>0,]
days_B<-days_B[rowSums(gbi_B)>0]
gbi_B<-gbi2_B

daysl_B[[j]]<-days_B
gbis_B[[j]]<-gbi_B

}

#We now go through and assign a location to every subgroup
sglocs_B<-list()
for(i in 1:n_groups_B){
  tx_B<-rep(NA,dim(gbis_B[[i]])[1])
  ty_B<-rep(NA,dim(gbis_B[[i]])[1])
  sglocs_B[[i]]<-data.frame(tx_B,ty_B)
  names(sglocs_B[[i]])<-c("x","y")
  sglocs_B[[i]]$x<-group_locs_B[i,1]+round(rnorm(dim(gbis_B[[i]])[1],0,2))
  sglocs_B[[i]]$y<-group_locs_B[i,2]+round(rnorm(dim(gbis_B[[i]])[1],0,2))
}

#Vector recording number of individuals in each group
n_inds_B<-numeric()
for(i in 1:n_groups_B){
  n_inds_B[i]<-dim(gbis_B[[i]])[2]
}

#Calculate total individuals in the population
n_tot_B<-sum(n_inds_B)

#Population-level individual identities
inds_tot_B<-seq(1,n_tot_B,1)

#Information on each individual's group membership
g_tot_B<-rep(seq(1,n_groups_B,1),n_inds_B)

#Information on each individual's within-group identity
gi_tot_B<-seq(1,n_inds_B[1],1)
for(i in 2:n_groups_B){
  gi_tot_B<-c(gi_tot_B,seq(1,n_inds_B[i],1))
}

#We now calculate the full population association network
full_net_B<-matrix(0,nr=n_tot_B,nc=n_tot_B)

#Counts up between-group associations
for(i in 1:100){
  for(j in 1:(n_groups_B-1)){
    for(k in (j+1):n_groups_B){
      tA_B<-paste0(sglocs_B[[j]][,1],"-",sglocs_B[[j]][,2])
      tB_B<-paste0(sglocs_B[[k]][,1],"-",sglocs_B[[k]][,2])
      tA2_B<-tA_B[daysl_B[[j]]==i]
      tB2_B<-tB_B[daysl_B[[k]]==i]
      tt_B<-match(tA2_B,tB2_B)
      if(sum(is.na(tt_B))<length(tt_B)){
      if(group_clans_B[j]==group_clans_B[k]){same<-rbinom(1,1,p_wc_B)}
```

```
      if(group_clans_B[j]!=group_clans_B[k]){same<-rbinom(1,1,p_bc_B)}
      if(same==1){
        paste(i,j,k)
        for(m in length(tt_B)){
          if(is.na(tt_B[m])==FALSE){
            tsg1_B<-which(tA_B==tA2_B[m]&daysl_B[[j]]==i)
            tsg2_B<-which(tB_B==tB2_B[tt_B[m]]&daysl_B[[k]]==i)
            tid1_B<-which(gbis_B[[j]][tsg1_B,]==1)
            tid2_B<-which(gbis_B[[k]][tsg2_B,]==1)
            tid1a_B<-inds_tot_B[g_tot_B==j&gi_tot_B%in%tid1_B]
            tid2a_B<-inds_tot_B[g_tot_B==k&gi_tot_B%in%tid2_B]
            full_net_B[tid1a_B,tid2a_B]<-full_net_B[tid1a_B,tid2a_B]+1
            full_net_B[tid2a_B,tid1a_B]<-full_net_B[tid1a_B,tid2a_B]
          }
        }
      }
      }
    }
  }
}

#converts between group associations to SRIs
for(i in 1:(nrow(full_net_B)-1)){
  for(j in (i+1):nrow(full_net_B)){
    full_net_B[i,j]<-full_net_B[i,j]/(200-full_net_B[i,j])
    full_net_B[j,i]<-full_net_B[i,j]
  }
}

#Adds within-group associations to the population network
for(i in 1:n_groups_B){
  full_net_B[inds_tot_B[g_tot_B==i],inds_tot_B[g_tot_B==i]]<-get_network2(gbis_B
[[i]])
}

#Plots the population social network
full_net2_B<-graph.adjacency(full_net_B,mode="undirected",weighted=TRUE)
plot(full_net2_B,vertex.color=unlist(noses_B),vertex.label=NA,vertex.size=4,edge.w
idth=(edge_attr(full_net2_B)$weight*8)^2,main="Population association network for
 collaborator")
```



Section 2.5 – Summary for Section 2

Burbils then (coincidentally, of course) are a study system in which social network analysis offers a perfect tool to answer key questions about social behaviour and ecology

Section 3

Social network analysis examples

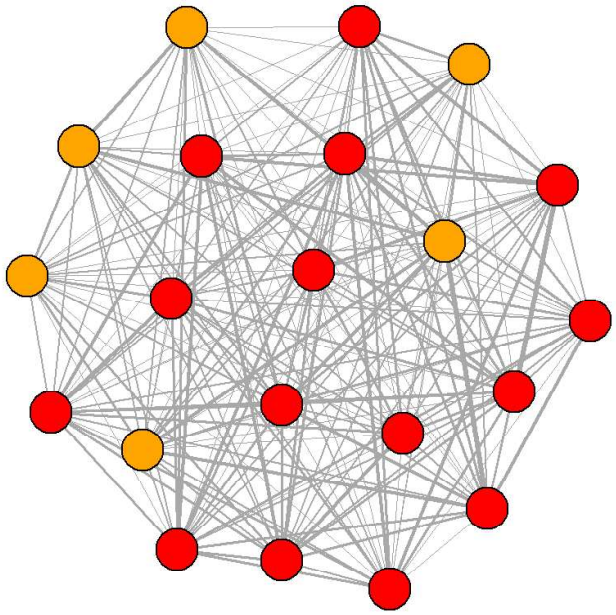
Section 3.1 – PERMUTATION-BASED REFERENCE MODELS

Section 3.1.1 – Comparing two approaches to using reference models

Our first analyses are for the examples presented in Table 1, with two research groups asking questions about the associations of Burbils in our first burbil group.

Team 1 ask a specific research question: do burbils associate by nose colour?

```
#First we extract the association network for team 1 from the group-by-individual
 matrix using the asnipe package
MAT1<-get_network2(gbis[[1]])

#We can the plot the network
NET1<-graph.adjacency(MAT1,mode="undirected",weighted=TRUE)
plot(NET1,vertex.label=NA,vertex.color=noses[[1]],edge.width=(edge_attr(NET1)$weig
ht*8)^2)
```

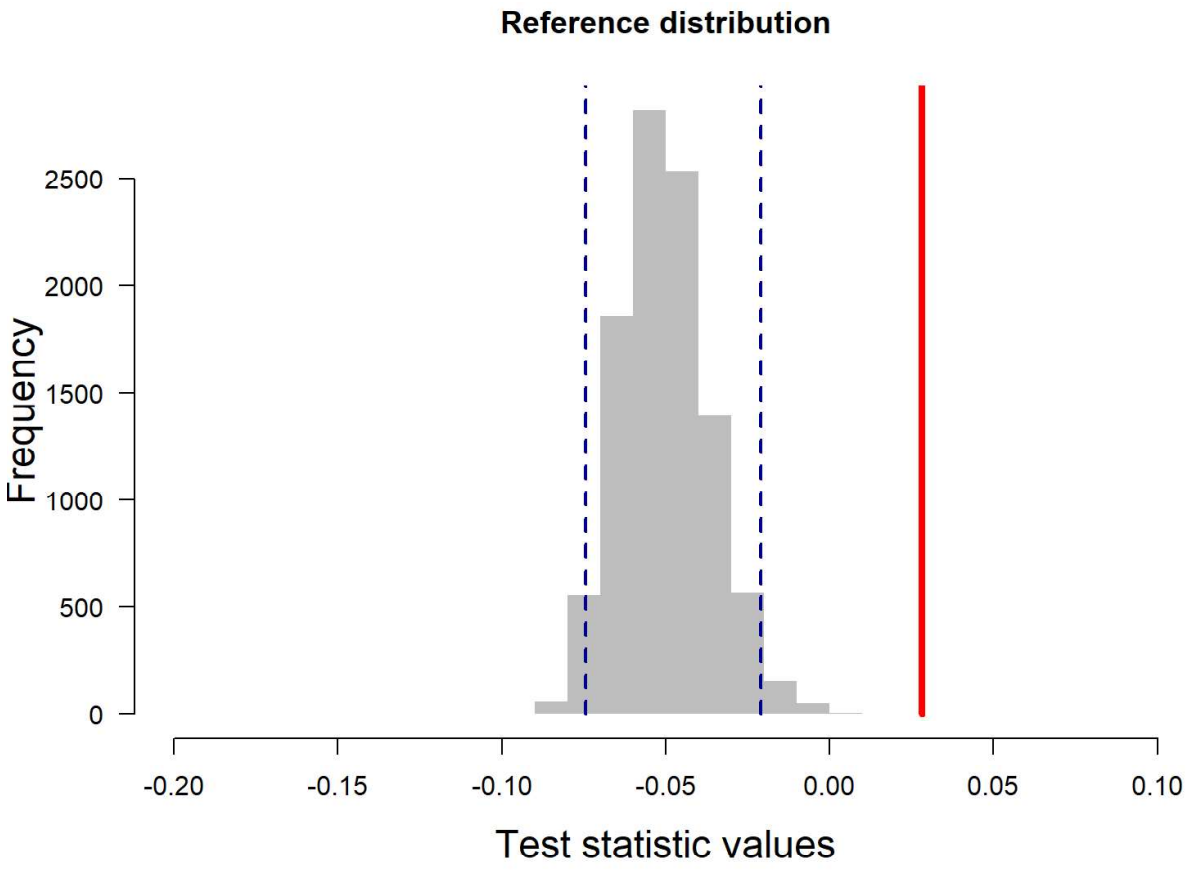


```
#We now calculate assortativity by nose colour in the real network
obs<-assortnet::assortment.discrete(MAT1, types=noses[[1]], weighted = TRUE, SE =
FALSE, M = 1)

#We then use node swap permutations to generate a reference distribution for assor
tativity
#We choose this type of permutation to break the correlation between nose colour a
nd network position
reference<-numeric()
MAT_T<-sna::rmperm(MAT1)
for(i in 1:9999){
reference[i]<-assortnet::assortment.discrete(MAT_T,types=noses[[1]], weighted = TR
UE, SE = FALSE, M = 1)$r
MAT_T<-sna::rmperm(MAT_T)
}
```

```
#We then add the observed assortativity to the reference distribution
reference2<-c(obs$r,reference)

#We can then calculate a p value by comparing the observed assortativity to the re
ference dataset. We are using a two-tailed test. Therefore, if we assume alpha=0.0
5 then assortativity is different to that expected by chance when p<0.025 (greater
than chance) or p>0.975 (less than chance)
sum(obs$r<reference2)/length(reference2)
```

```
## [1] 0
```

```
#Here we produce a plot to show this result. The grey histogram is the reference d
ataset, the blue dashed lines the 2.5% and 97.5% quantiles of the reference datase
t and the red line is the observed assortativity
par(xpd=FALSE)
hist(reference,las=1,xlim=c(-0.2,0.1),col="grey",border=NA,main="Reference distrib
ution",xlab="Test statistic values",cex.lab=1.5,cex.axis=1)
lines(x=c(obs$r,obs$r),y=c(0,5000),col="red",lwd=4)
lines(x=rep(quantile(reference2,0.025),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
lines(x=rep(quantile(reference2,0.975),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
```

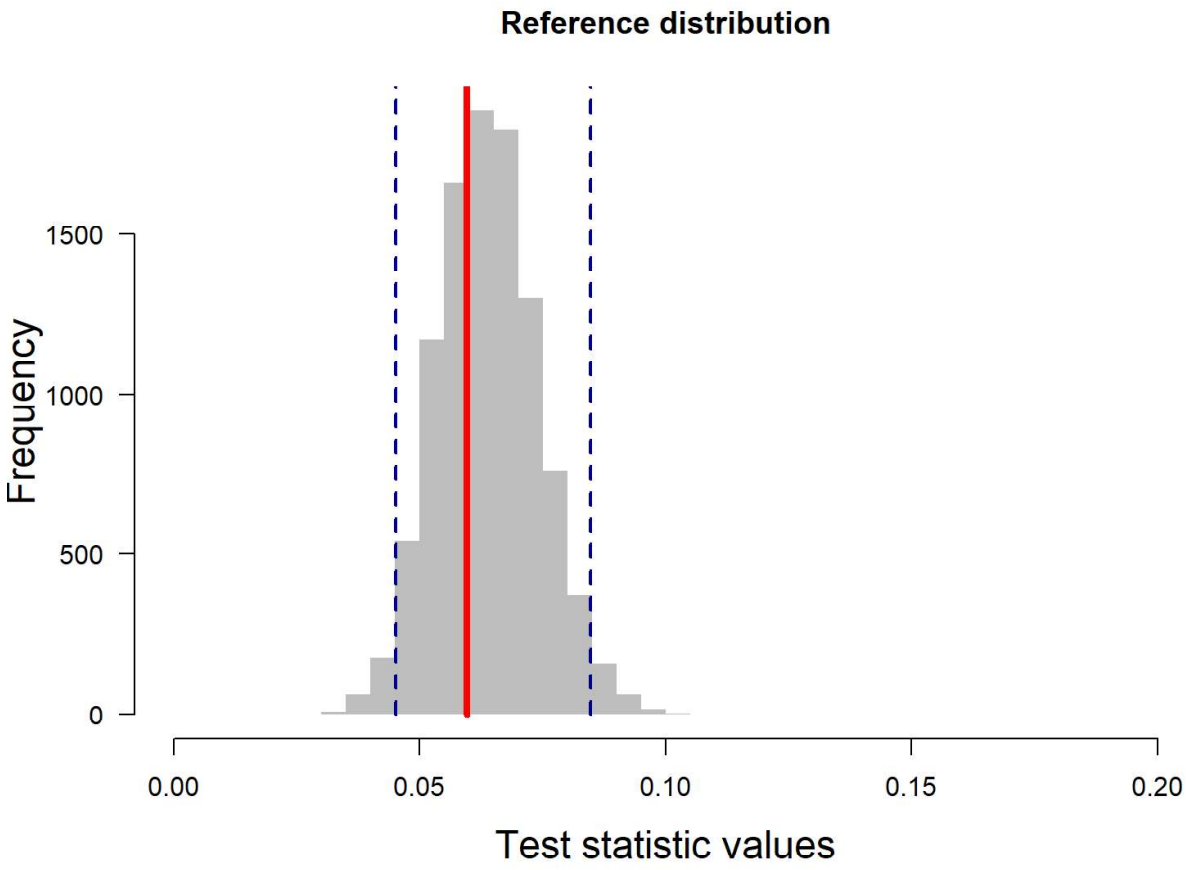


Team 2 ask a rather vague question (do burbils associate at random?) and are less careful when designing and implementing their reference model.

```
#First we extract the association network for team 2 from the group-by-individual
 matrix using the asnipe package
MAT1<-get_network2(gbis[[1]])

#Calculate the coefficient of variation in weighted degree (naively)
obs<-sd(rowSums(MAT1))/mean(rowSums(MAT1))

#Generate reference model using a random graph and edge weight distribution
reference<-numeric()
for(i in 1:9999){
  net_r<-igraph::erdos.renyi.game(n=nrow(MAT1),p.or.m=sum(sign(MAT1))/2,type="gnm"
)
net_r<-set_edge_attr(net_r,"weight",value=rnorm(n=sum(sign(MAT1))/2,mean=mean(MAT
1),sd=sd(MAT1)))
mat_r<-as_adjacency_matrix(net_r,type="both",attr="weight",sparse=FALSE)
diag(mat_r)<-0
  reference[i]<-sd(rowSums(mat_r))/mean(rowSums(MAT1))
}

#We then add the observed coefficient of variation to the reference distribution
reference2<-c(obs,reference)

#Calculate p value
sum(obs<reference2)/length(reference2)
```

```
## [1] 0.656
```

```
#Plot randomisation result
par(xpd=FALSE)
hist(reference,las=1,xlim=c(0,0.2),col="grey",border=NA,main="Reference distributi
on",xlab="Test statistic values",cex.lab=1.5,cex.axis=1)
lines(x=c(obs,obs),y=c(0,5000),col="red",lwd=4)
lines(x=rep(quantile(reference2,0.025),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
lines(x=rep(quantile(reference2,0.975),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
```

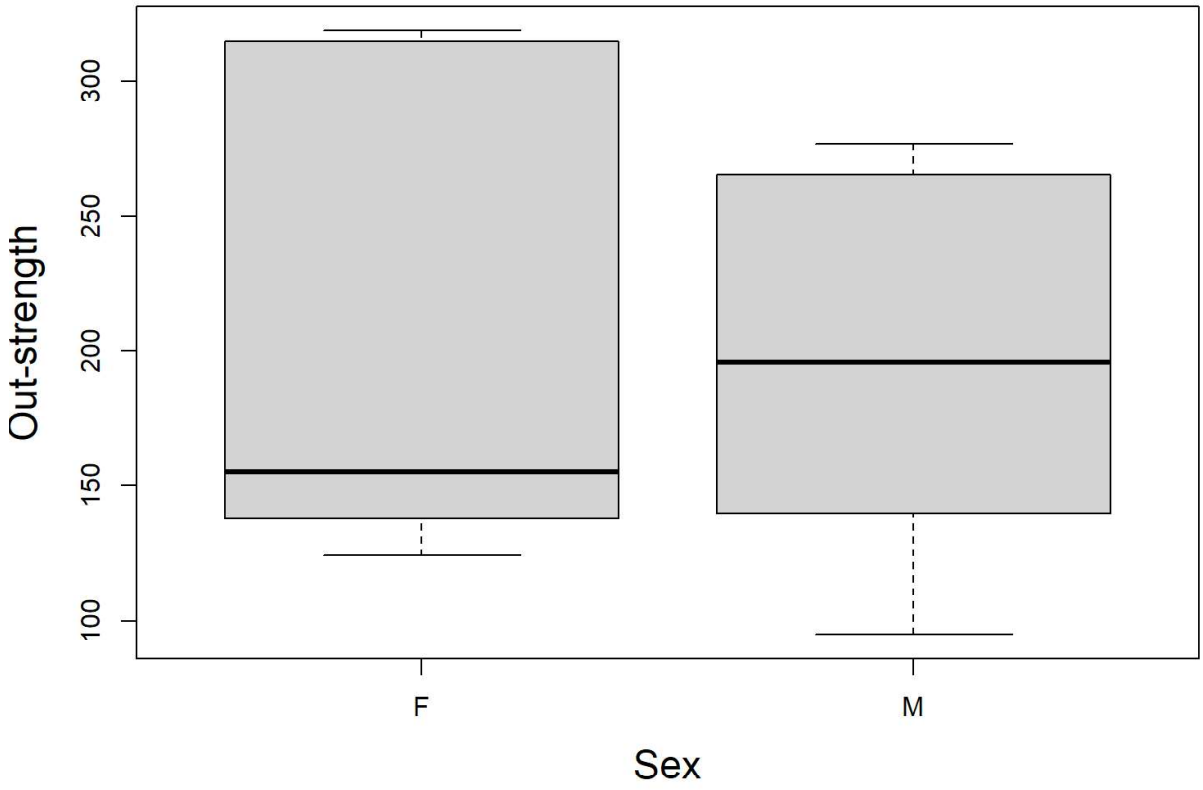


Section 3.1.2 – Node feature swaps

Node swaps can be used to test a range of hypotheses and network types, for example they are also appropriate to test statistical significance of regression models when a network measure is the response variable.

Here we test the relationship between sex (female *versus* male) and weighted degree.

```
#First we plot the relationship
boxplot(strength(dom_net,mode="out")~sexes[[1]],xlab="Sex",ylab="Out-strength",ce
x.lab=1.5)
```

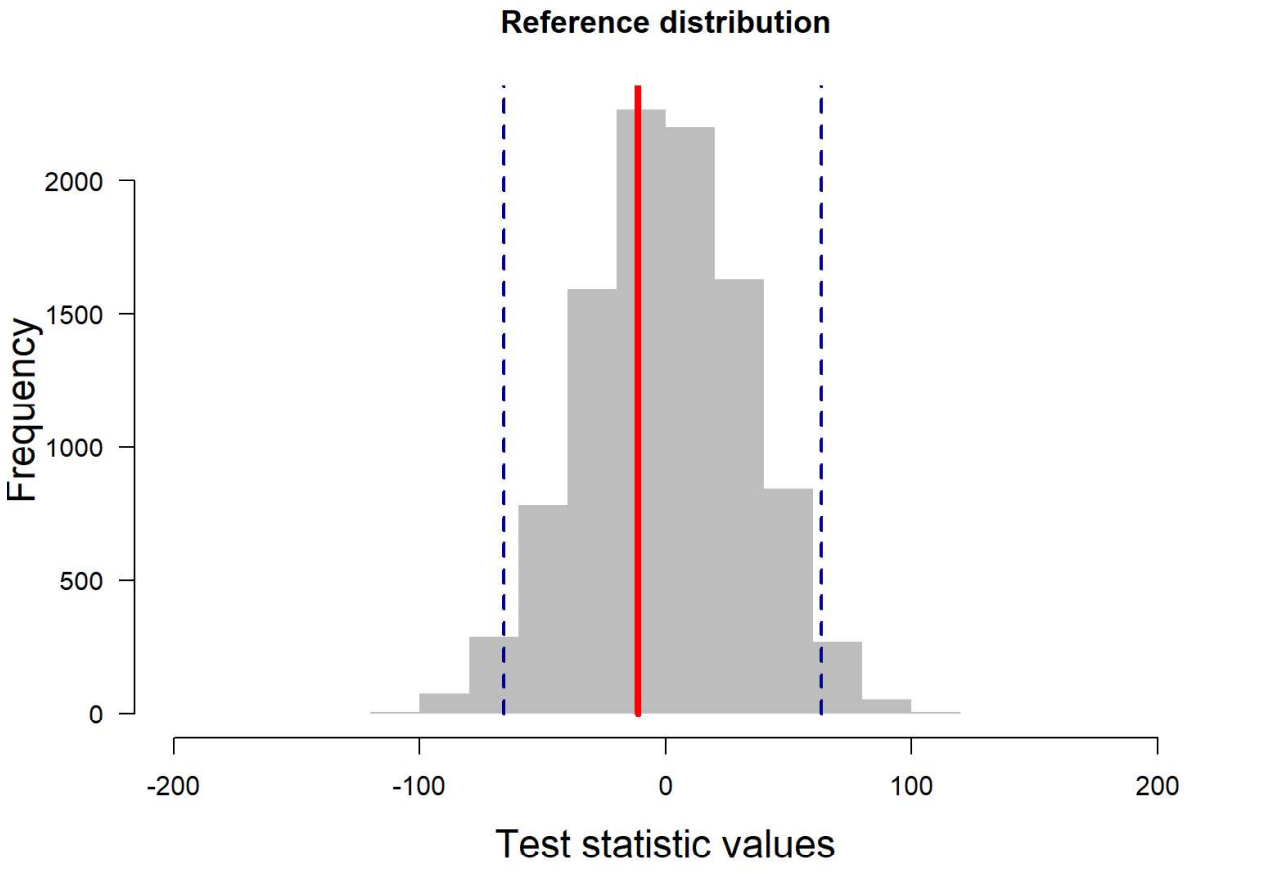



```
#We then convert our dominance network into an adjacency matrix form
MAT_DOM<-as_adjacency_matrix(dom_net,sparse=FALSE,attr="weight")

#We can the calculate the weighted out-degree for all individuals in the dominance
network
str_obs<-rowSums(MAT_DOM)

#We then choose our test statistic. We select the coefficient for effect of sex on
weighted degree estimated from a linear model. We calculate this for the observed
 network here
obs<-coef(lm(str_obs~sexes[[1]]))[2]

#We then use node swap permutations to generate a reference distribution for the n
ull relationship. Our node swaps (as above) using the rmperm function in the R pac
kage sna.
reference<-numeric()
MAT_T<-sna::rmperm(MAT_DOM)
for(i in 1:9999){
str_perm<-rowSums(MAT_T)
reference[i]<-coef(lm(str_perm~sexes[[1]]))[2]
MAT_T<-sna::rmperm(MAT_T)
}

#We then add the observed coefficient to the reference distribution
reference2<-c(obs,reference)

#We can then calculate a p value by comparing the observed linear relationship to
 those in the reference dataset. We are using a two-tailed test. Therefore, if we
```

```
 assume alpha=0.05 then assortativity is different to that expected by chance when
p<0.025 (weighted out-degree of males higher than females) or p>0.975 (weighted ou
t-degree of males less than females)
sum(obs<reference2)/length(reference2)
```

```
## [1] 0.638
```

```
#Here we produce a plot to show this result. The grey histogram is the reference d
ataset, the blue dashed lines the 2.5% and 97.5% quantiles of the reference datase
t and the red line is the observed relationship
par(xpd=FALSE)
hist(reference,las=1,xlim=c(-200,200),col="grey",border=NA,main="Reference distrib
ution",xlab="Test statistic values",cex.lab=1.5)
lines(x=c(obs,obs),y=c(0,5000),col="red",lwd=4)
lines(x=rep(quantile(reference2,0.025),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
lines(x=rep(quantile(reference2,0.975),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
```



An important thing to bear in mind with node swaps, especially unconstrained node swaps as we have used above, is that they can't control for structure in the network. This could lead to potentially misleading conclusions being drawn when: (a) there are biological processes operating at different scales that might be important in driving observed patterns; or (b) there is variation in sampling intensity. For example, in the example above we know (because we simulated the data) that there is an equal probability of males and females being observed. However, if one sex was more likely to be observed than the other, it would be expected to have a higher weighted degree, but this would be driven by sampling bias and not biology. It is

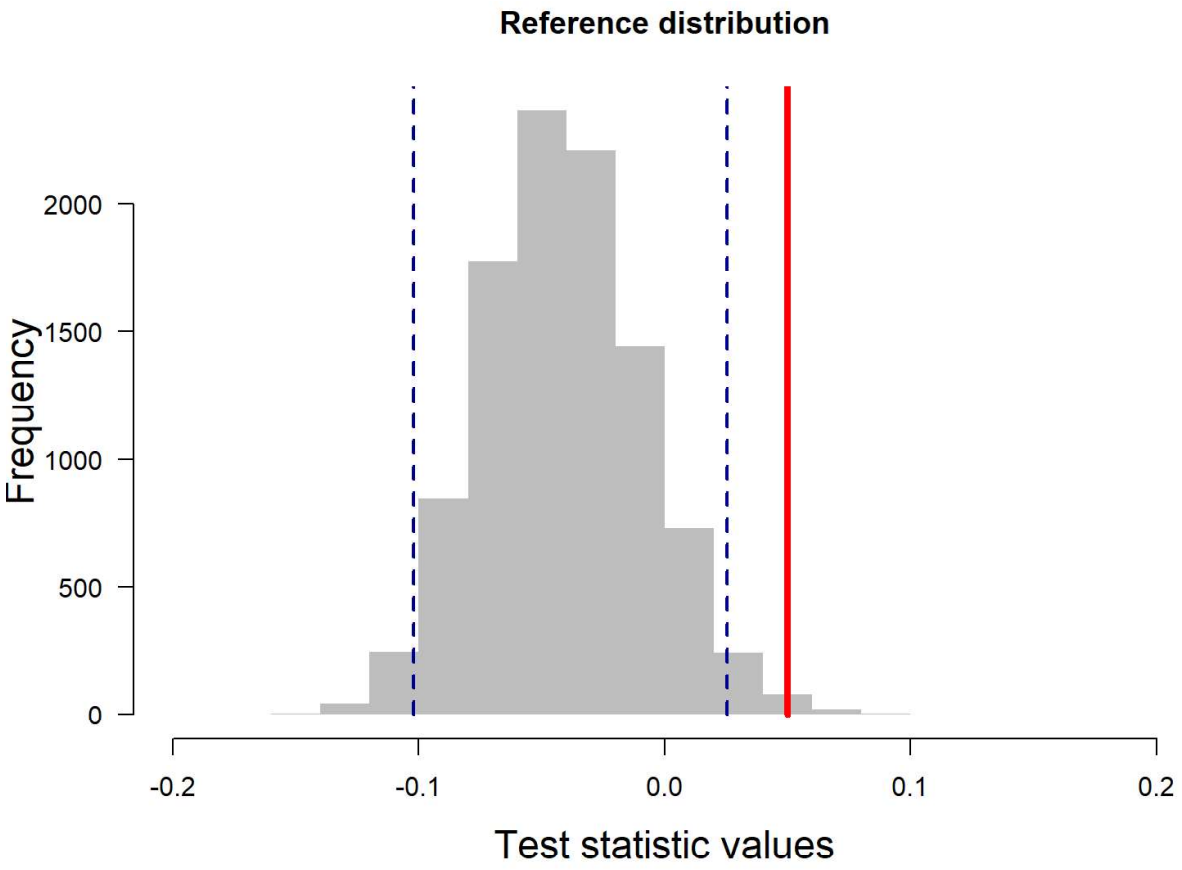
hard to deal with this directly using node feature permutations, and they would need to be combined with other methodologies.

We provide an example of (a) here. We know that the burbil association network is assorted by nose colour. Therefore, we want to know if the network of affiliative interactions is too.

```
#Convert affiliative network into an adjacency matrix
MAT_AFF<-as_adjacency_matrix(aff_net,sparse=FALSE,attr="weight")

#Calculate the observed assortativity of the affiliative network
obs<-assortnet::assortment.discrete(MAT_AFF, types=noses[[1]], weighted = TRUE, SE
= FALSE, M = 1)$r

#Generate the reference distribution
reference<-numeric()
MAT_T<-sna::rmperm(MAT_AFF)
for(i in 1:9999){
reference[i]<-assortnet::assortment.discrete(MAT_T,types=noses[[1]], weighted = TR
UE, SE = FALSE, M = 1)$r
MAT_T<-sna::rmperm(MAT_T)
}

#Add the observed assortativity to the reference dataset
reference2<-c(obs,reference)

#Calculate the p value. (p<0.025 would equate to the network being positively asso
rted by nose colour and p>0.975 to the network being negatively assorted by nose c
olour)
sum(obs<reference2)/length(reference2)
```

```
## [1] 0.0048
```

```
#We can then plot the result as we have done above
par(xpd=FALSE)
hist(reference,las=1,xlim=c(-0.2,0.2),col="grey",border=NA,main="Reference distrib
ution",xlab="Test statistic values",cex.lab=1.5)
lines(x=c(obs,obs),y=c(0,5000),col="red",lwd=4)
lines(x=rep(quantile(reference2,0.025),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
lines(x=rep(quantile(reference2,0.975),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
```



We find that the affiliative network is assorted by nose colour. However, we haven't controlled for association network structure in our reference model and we know that this places important constraints on the opportunities to interact. This is a key consideration when interpreting the results of simple reference models like this. We revisit this example later on.

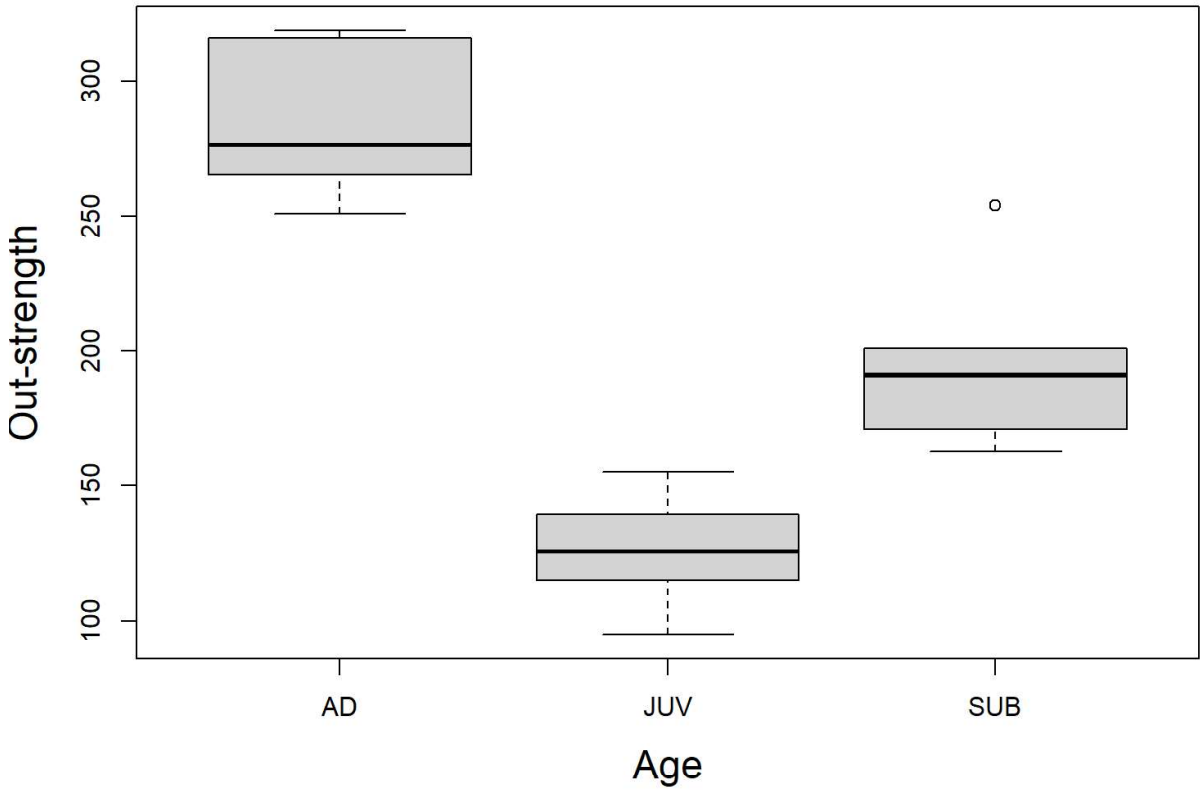
Section 3.1.3 – Edge feature swaps

As described in the main text, we don't just have to swap nodes. To test some hypotheses in directed networks, permuting the direction of edges can be a useful way to generate a reference distribution.

Here we provide an example of swapping edge directions. We test the hypothesis that adults tend to initiate more dominance interactions than younger individuals.

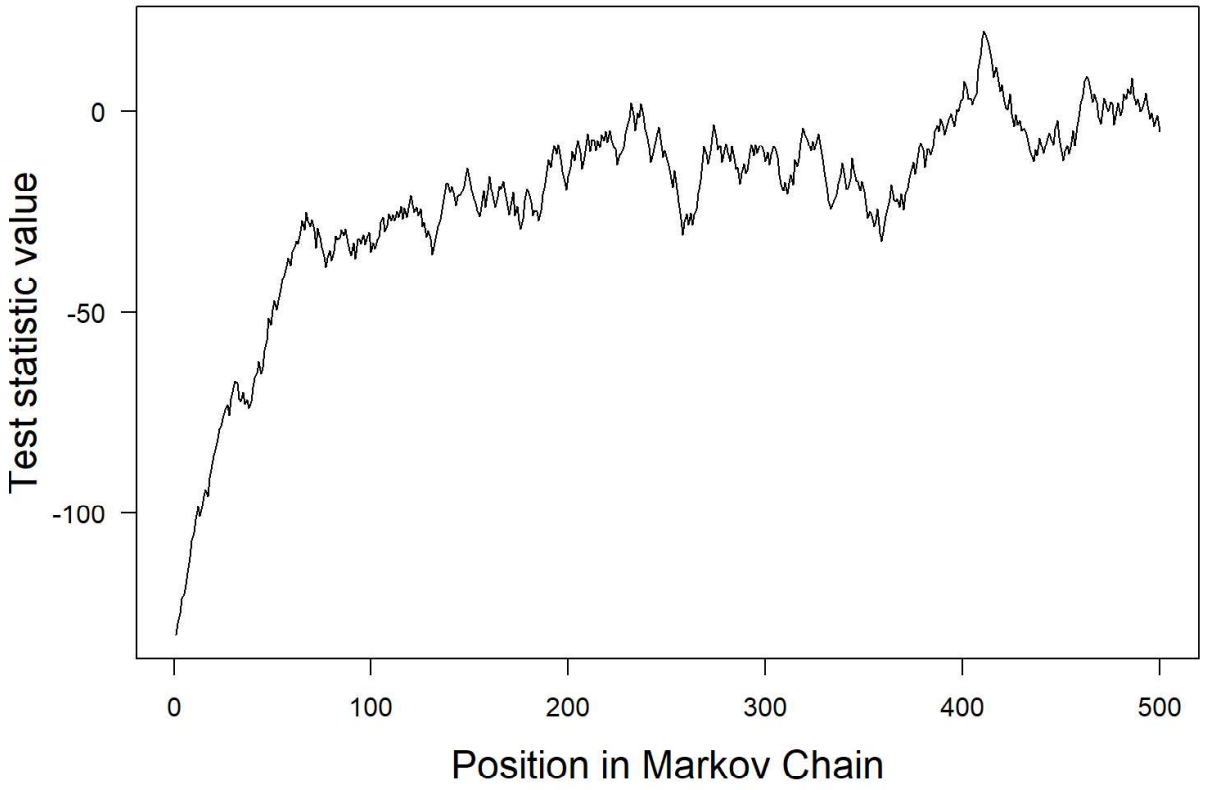
So far all of our permutations have randomised the whole network in one go. Now we move on to a type of permutation in which we make a single swap at a time (the direction of one edge) and these swaps occur successively causing the 'permutedness' of the network to increase until it is a uniform sample of the reference distribution. We are generating what is known as a Markov Chain, and sampling from it

```
#First we check the relationship
boxplot(strength(dom_net,mode="out")~ages[[1]],ylab="Out-strength",xlab="Age",cex.
lab=1.5)
```




```r
#Now we convert the network into an adjacency matrix
MAT_DOM<-as_adjacency_matrix(dom_net,sparse=FALSE,attr="weight")

#We now calculate the weighted out-degree (or out-strength) for all individuals in
the dominance network
str_obs<-strength(dom_net,mode="out")

#Here we collapse the "age" variable into a simple version with just adults (A) an
d youngsters (Y)
ageT<-ages[[1]]
ageT[ageT=="AD"]<-"A"
ageT[ageT=="SUB"|ageT=="JUV"]<-"Y"

#As per the previous example we choose a test statistic which is the coefficient o
f the linear model between weighted degree (response variable) and our simplified
 measure of age (explanatory variable)
obs<-coef(lm(str_obs~ageT))[2]

#We now generate the reference distribution
burnin<-numeric()
reference<-numeric()
MAT_T<-MAT_DOM

#Unlike for the node swaps we first conduct a period known as the burn-in, during
 which time the "permutedness" of the data gradually increases towards the referen
ce distribution. We plot this to show it
for(i in 1:500){
  tid1<-sample(which(ageT=="A"),1)
```

```
  tid2<-sample(which(ageT=="Y"),1)
  MAT_T2<-MAT_T
  MAT_T2[tid1,tid2]<-MAT_T[tid2,tid1]
  MAT_T2[tid2,tid1]<-MAT_T[tid1,tid2]
  MAT_T<-MAT_T2
  dn_r<-graph_from_adjacency_matrix(MAT_T,weighted=TRUE,mode="directed")
  str_ref<-strength(dn_r,mode="out")
  burnin[i]<-coef(lm(str_ref~ageT))[2]
}

#When we plot how the value of the test statistic changes as the number of swaps i
ncreases, we can see that it moves quickly away from the observed value and then a
fter ~100 swaps becomes relatively stable, or stationary.
plot(burnin,type="l",las=1,ylab="Test statistic value",xlab="Position in Markov Ch
ain",cex.lab=1.5)
```

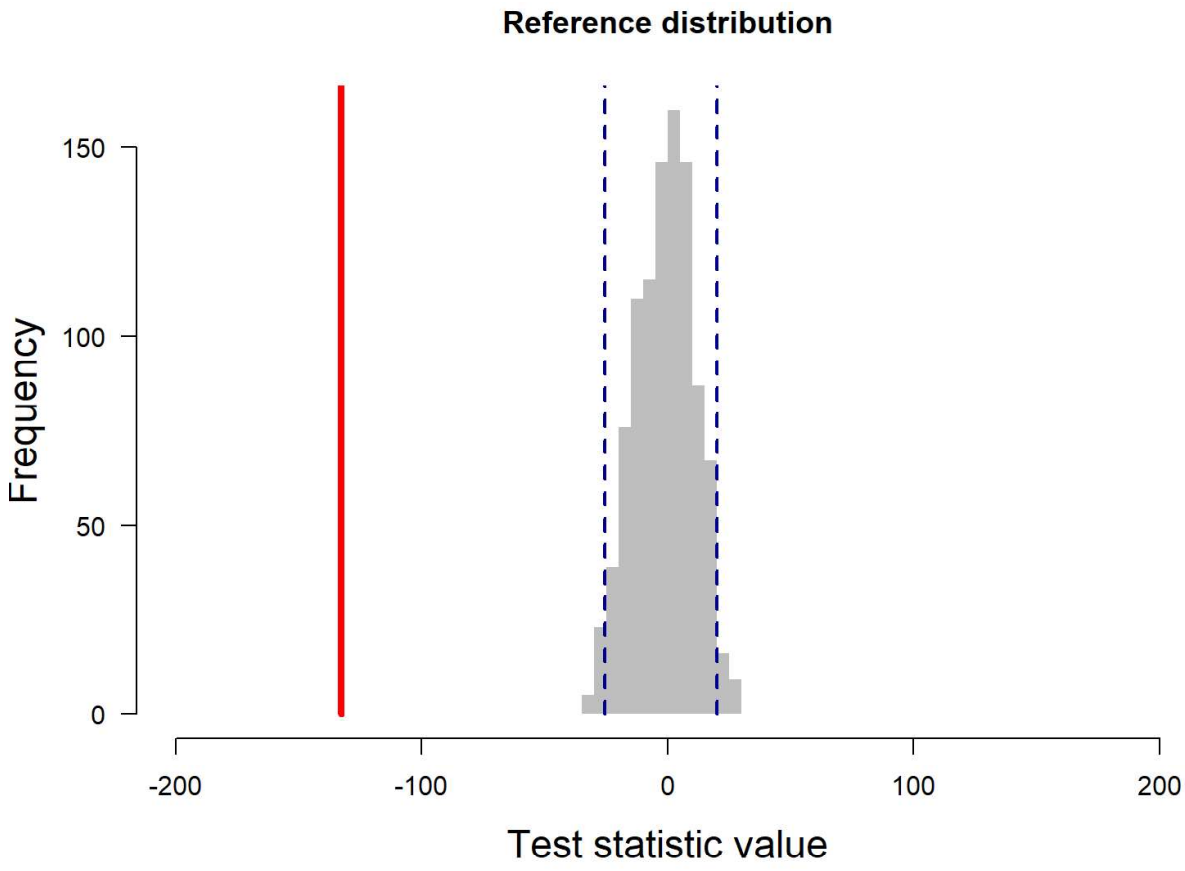



```
#We can then continue the Markov Chain and sample from it after each swap to gener
ate our reference distribution of test statistics.
for(i in 1:999){
  tid1<-sample(which(ageT=="A"),1)
  tid2<-sample(which(ageT=="Y"),1)
  MAT_T2<-MAT_T
  MAT_T2[tid1,tid2]<-MAT_T[tid2,tid1]
  MAT_T2[tid2,tid1]<-MAT_T[tid1,tid2]
  MAT_T<-MAT_T2
  dn_r<-graph_from_adjacency_matrix(MAT_T,weighted=TRUE,mode="directed")
  str_ref<-strength(dn_r,mode="out")
```

```
  reference[i]<-coef(lm(str_ref~ageT))[2]
}

#We then add the observed value to the reference distribution
reference2<-c(obs,reference)

#And calculate the p value (p<0.025 would equate to the youngsters having higher o
ut-strength in the dominance network and p>0.975 to youngsters having lower out-st
rength)
sum(obs<reference2)/length(reference2)
```

```
## [1] 0.999
```

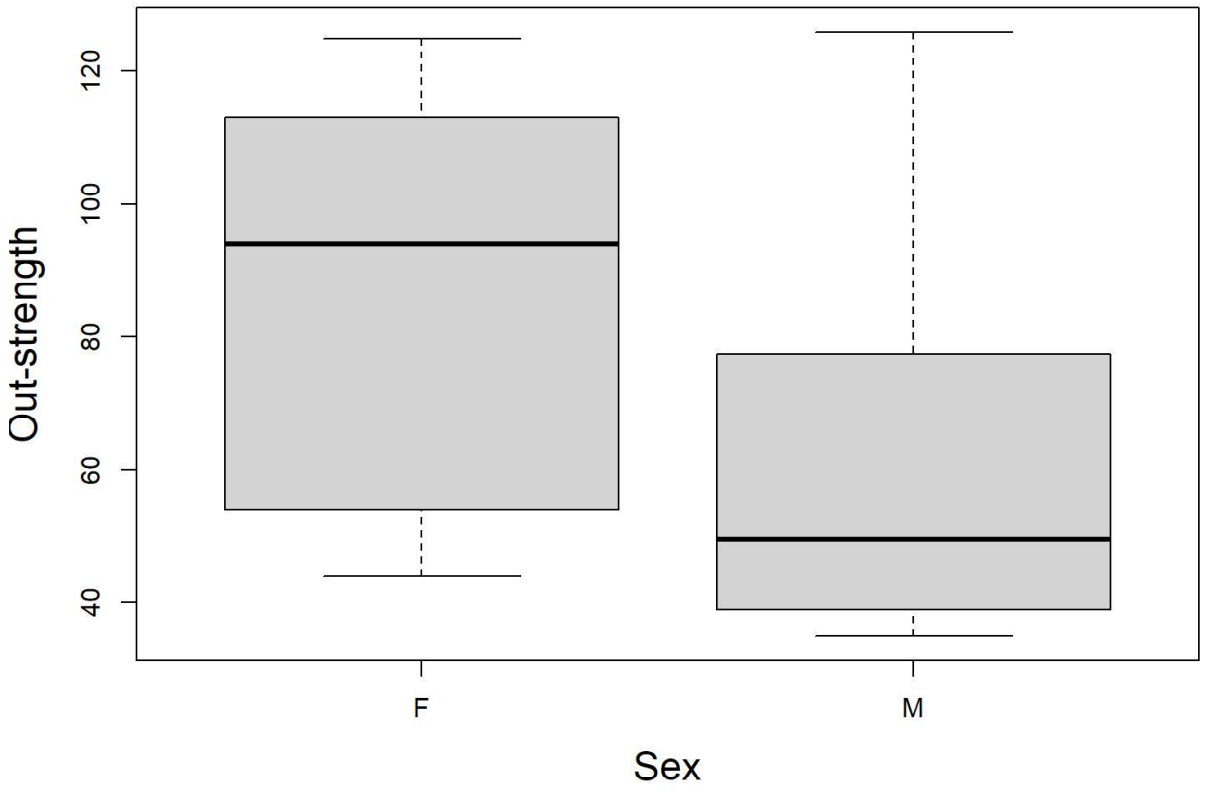
```
#We can then plot our results in the same way we have previously
par(xpd=FALSE)
hist(reference,las=1,xlim=c(-200,200),col="grey",border=NA,main="Reference distrib
ution",xlab="Test statistic value",cex.lab=1.5)
lines(x=c(obs,obs),y=c(0,5000),col="red",lwd=4)
lines(x=rep(quantile(reference2,0.025),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
lines(x=rep(quantile(reference2,0.975),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
```



As well as swapping the directions of edges, you could also swap edge weights to address some research questions, especially for well-connected networks. We provide a quick demonstration of how you could use edge weight swaps using our burbil affiliative networks here.

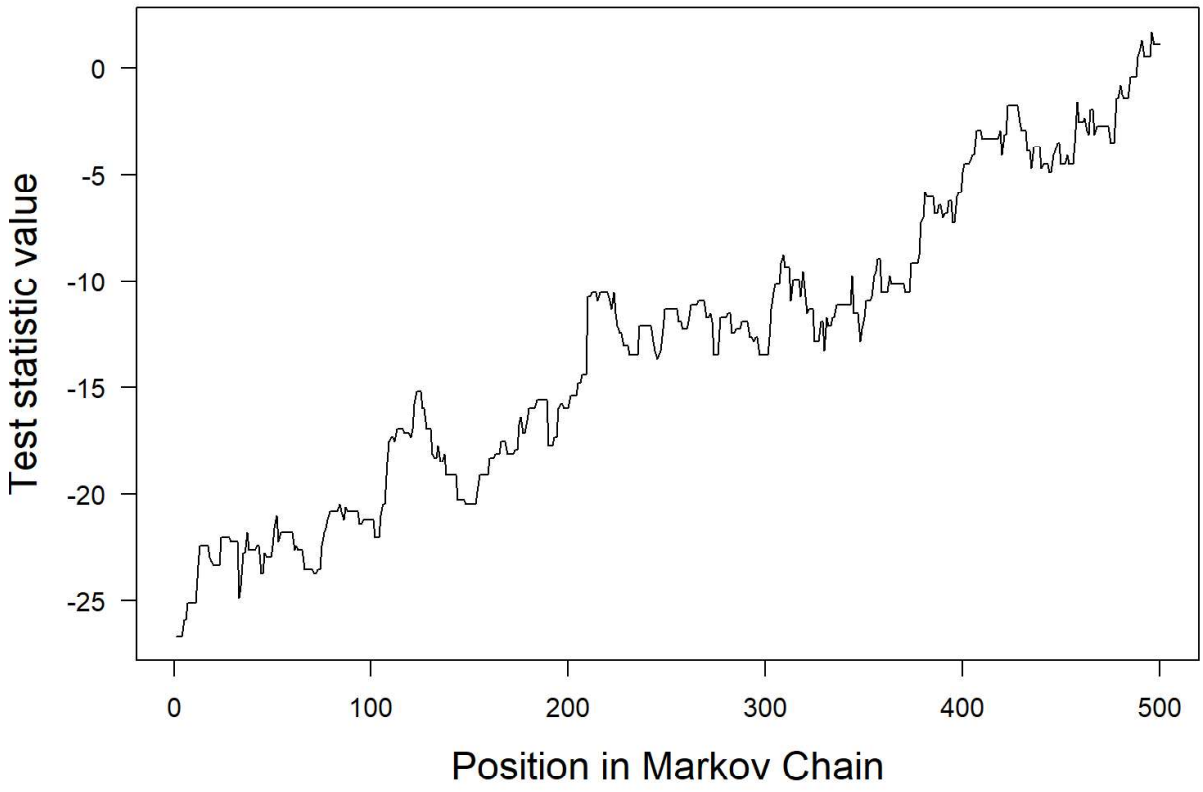
We test the hypothesis that sexes differ in their weighted degree in burbil networks of affiliative interactions. While we could create a reference model in other ways, in this case we choose to permute edge weights to randomise them with respect to sex.

```
#Convert affiliative network into an adjacency matrix
MAT_AFF<-as_adjacency_matrix(aff_net,sparse=FALSE,attr="weight")

#We now calculate the weighted out-degree (or out-strength) for all individuals in
the affiliative network
str_obs<-strength(aff_net,mode="out")

#Here we record the sexes of the individuals in the network
sexT<-sexes[[1]]

#Calculate the association between strength and sex in the observed affiliative ne
twork
obs<-coef(lm(str_obs~sexT))[2]

#First plot the relationship to investigate if any differences are apparent
boxplot(strength(aff_net,mode="out")~sexT,ylab="Out-strength",xlab="Sex",cex.lab=
1.5)
```




```
#The plot shows that, when we ignore other factors, males do initiate slightly mor
e affiliative interactions

#We now generate the reference distribution
burnin<-numeric()
```

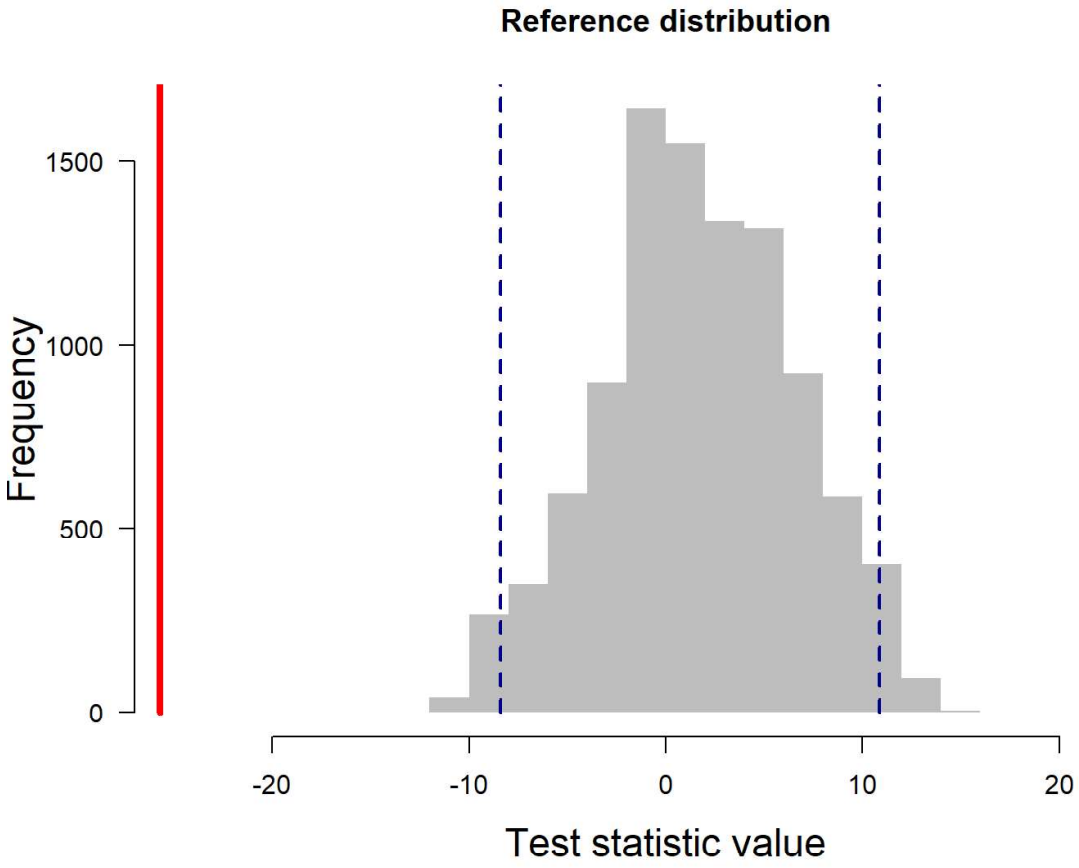
```
reference<-numeric()
MAT_T<-MAT_AFF

#Unlike for the node swaps we first conduct a period known as the burn-in, during
 which time the 'permutedness' of the data gradually increases towards the referen
ce distribution. We plot this to show it
for(i in 1:500){
  tid1<-sample(1:nrow(MAT_AFF),2,replace=F)
  tid2<-sample(1:nrow(MAT_AFF),2,replace=F)
  if(sum(tid1%in%tid2)!=2){
    MAT_T2<-MAT_T
    MAT_T2[tid1[1],tid1[2]]<-MAT_T[tid2[1],tid2[2]]
    MAT_T2[tid2[1],tid2[2]]<-MAT_T[tid1[1],tid1[2]]
    MAT_T<-MAT_T2
  }
  an_r<-graph_from_adjacency_matrix(MAT_T,weighted=TRUE,mode="directed")
  str_ref<-strength(an_r,mode="out")
  burnin[i]<-coef(lm(str_ref~sexT))[2]
}

#When we plot how the value of the test statistic changes as the number of swaps i
ncreases, we can see that it moves quickly away from the observed value and then a
fter ~100 swaps becomes relatively stable, or stationary.
plot(burnin,type="l",las=1,ylab="Test statistic value",xlab="Position in Markov Ch
ain",cex.lab=1.5)
```



```
#We can then continue the Markov Chain and sample from it after each swap to gener
ate our reference distribution of test statistics.
for(i in 1:9999){
  tid1<-sample(1:nrow(MAT_AFF),2,replace=F)
  tid2<-sample(1:nrow(MAT_AFF),2,replace=F)
  if(sum(tid1%in%tid2)!=2){
    MAT_T2<-MAT_T
    MAT_T2[tid1[1],tid1[2]]<-MAT_T[tid2[1],tid2[2]]
    MAT_T2[tid2[1],tid2[2]]<-MAT_T[tid1[1],tid1[2]]
    MAT_T<-MAT_T2
  }
  an_r<-graph_from_adjacency_matrix(MAT_T,weighted=TRUE,mode="directed")
  str_ref<-strength(an_r,mode="out")
  reference[i]<-coef(lm(str_ref~sexT))[2]
}

#We then add the observed value to the reference distribution
reference2<-c(obs,reference)

#And calculate the p value (p<0.025 would equate to the males having higher out-st
rength in the dominance network and p>0.975 to males having lower out-strength)
sum(obs<reference2)/length(reference2)
```

```
## [1] 0.9999
```

```
#We can then plot our results in the same way we have previously
par(xpd=FALSE)
hist(reference,las=1,xlim=c(-25,25),col="grey",border=NA,main="Reference distribut
ion",xlab="Test statistic value",cex.lab=1.5)
lines(x=c(obs,obs),y=c(0,5000),col="red",lwd=4)
lines(x=rep(quantile(reference2,0.025),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
lines(x=rep(quantile(reference2,0.975),2),y=c(0,5000),col="darkblue",lwd=2,lty=2)
```




```
#We interpret this result as showing that females tend to have higher out-strength than males.

#However, when we evaluate our analysis we consider the role of other variables, especially as our group of burbils is relatively small. So we inspect the data we have on individual traits and realise that more juveniles in this group happen to be female

sum(sexes[[1]]=="F"&ages[[1]]=="JUV")
```

```
## [1] 5
```

```
sum(sexes[[1]]=="M"&ages[[1]]=="JUV")
```

```
## [1] 3
```

```
#A better approach would be to include age in our reference model (more tightly control the permutations). This example also shows the challenge of using network analyses in small groups like this, even with plentiful data on the interactions themselves.
```

Section 3.1.4 – Raw data swaps

We can also use permutations to make swaps in the raw data used to generate the network. It can be helpful to think of these as networks in themselves. The raw data used to construct animal social networks tends to come in two forms:

- Group-by-individual (GBI) matrices: these are effectively bipartite networks in which individuals are connected to particular grouping events. GBI matrices are used to generate association networks by collapsing this bipartite network using the assumption that individuals within each grouping event are connected
- Edge lists: these list the initiator and receiver of a set of behavioural interactions (or can also be used for contacts detected using proximity loggers). They are edge lists for a multigraph (i.e. a network with multiple rather than weighted edges) comprising the same set of individuals.

Permutations such as this are often called **datastream permutations**.

Section 3.1.4.1 – Permutations applied to group-by-individual matrices

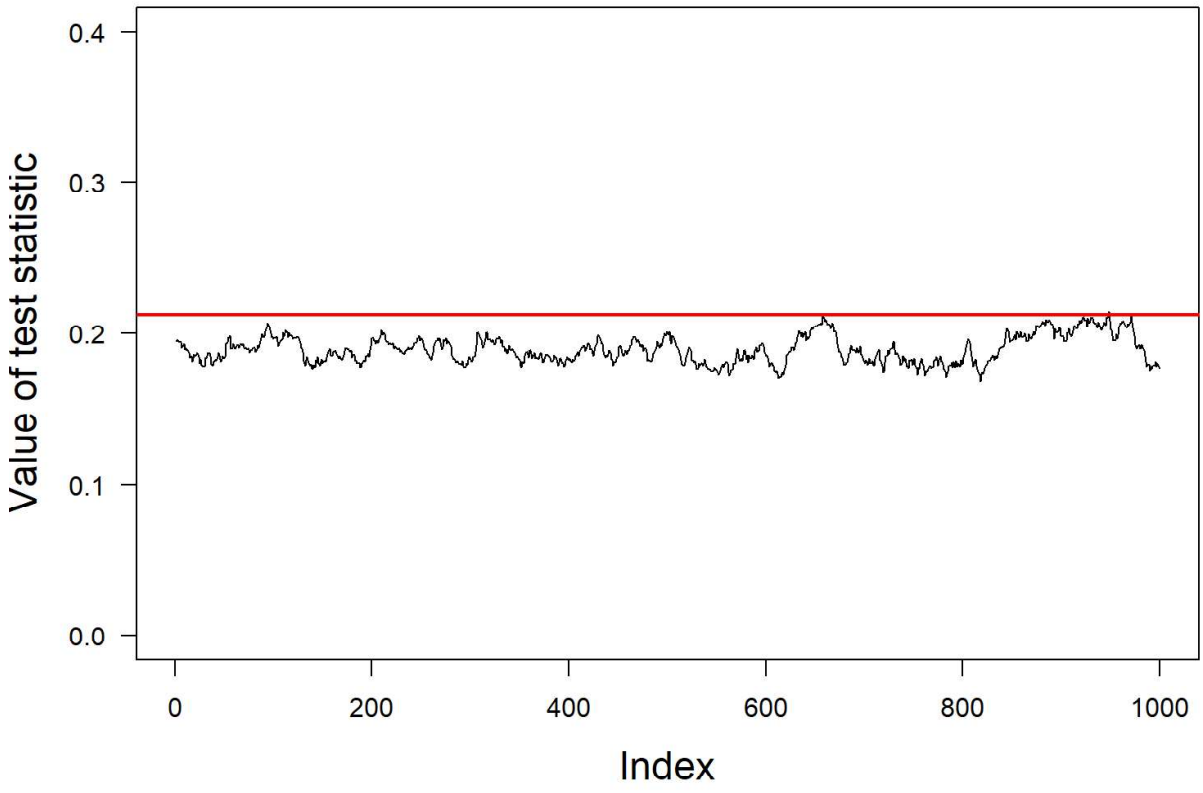
We provide an example of datastream permutations for a GBI here. We are going to test the hypothesis that our association network within a single group is different from random. We first compare it to a completely randomised network but then realise that this isn't especially interesting as we already know that the network is assorted by nose colour. Therefore, we test a second hypothesis that associations within a group are random once we have accounted for the assortativity by nose colour.

This helps us show how additional constraints can be added to these datastream permutations, and also highlights the value of constructing multiple reference models to have a good understanding of your data.

```
#Define some functions that we will use to calculate our test statistic
CoV<-function(a){
  a2<-a
  diag(a2)<-NA
  return(sd(a2,na.rm=T)/mean(a2,na.rm=T))
}
CoV2<-function(a){
  return(sd(a[a!=0])/mean(a[a!=0]))
}

#We choose the coefficient of variation in edge weight to be our test statistic. T
his is often used to test whether networks are different to the random expectation
within particular constraints
obs_cv<-CoV(get_network2(gbi))


#We first store the GBI and relevant information on which days groups were observe
d. We use the latter to constrain our permutations
gbi<-gbis[[1]]
day<-daysl[[1]]

## We then generate our reference distribution
#We constrain our swaps so that they have to occur between two groups occurring on
the same day.
#We have additional constraints that stop individuals being swapped into a group t
hey already occur in (this shouldn't matter here, but does happen if an individual
```

```
can be recorded multiple times within the time period that swaps are constrained t
o occur within)
#Note that if we try to swap an individual into a group that it already occurs in
 and reject the swap then we keep the current version of the permuted GBI for the
 next step of the Markov Chain rather than simply trying again. This is important
 to ensure uniform sampling of the reference distribution.
gbi_t<-gbi
rgbis<-list()

#As above we have a burn-in period for the Markov Chain
for(i in 1:500){
  #sample an individual/grouping-event
  pind<-which(gbi_t>0,arr.ind=TRUE)
  tind1<-pind[sample(1:nrow(pind),1),]
  #record the day on which that individual/grouping-event occurred
  td<-which(day==day[tind1[1]])
  #sample a second individual/grouping-event that occurs on the same day
  pind2<-pind[which(pind[,1]%in%td),]
  tind2<-pind2[sample(1:nrow(pind2),1),]
  #If additional constraints are met then conduct swap
  if(tind1[1]!=tind2[1]&tind1[2]!=tind2[2]){
    if(gbi_t[tind1[1],tind2[2]]==0&gbi_t[tind2[1],tind1[2]]==0){
        gbi_t2<-gbi_t
        gbi_t2[tind2[1],tind1[2]]<-gbi_t[tind1[1],tind1[2]]
        gbi_t2[tind1[1],tind1[2]]<-gbi_t[tind2[1],tind1[2]]
        gbi_t2[tind1[1],tind2[2]]<-gbi_t[tind2[1],tind2[2]]
        gbi_t2[tind2[1],tind2[2]]<-gbi_t[tind1[1],tind2[2]]
        gbi_t<-gbi_t2
    }
  }
}

#We can then continue the Markov Chain and sample from it to generate our referenc
e distribution of test statistics. Here we conduct 10000 swaps but we only save ev
ery 10 iterations (known as a thinning interval) to avoid auto-correlation that ma
y occur because of rejected swaps
c<-1
for(i in 1:10000){
  pind<-which(gbi_t>0,arr.ind=TRUE)
  tind1<-pind[sample(1:nrow(pind),1),]
  td<-which(day==day[tind1[1]])
  pind2<-pind[which(pind[,1]%in%td),]
  tind2<-pind2[sample(1:nrow(pind2),1),]
  if(tind1[1]!=tind2[1]&tind1[2]!=tind2[2]){
    if(gbi_t[tind1[1],tind2[2]]==0&gbi_t[tind2[1],tind1[2]]==0){
        gbi_t2<-gbi_t
        gbi_t2[tind2[1],tind1[2]]<-gbi_t[tind1[1],tind1[2]]
        gbi_t2[tind1[1],tind1[2]]<-gbi_t[tind2[1],tind1[2]]
        gbi_t2[tind1[1],tind2[2]]<-gbi_t[tind2[1],tind2[2]]
        gbi_t2[tind2[1],tind2[2]]<-gbi_t[tind1[1],tind2[2]]
        gbi_t<-gbi_t2
    }
  }
  #This is where we save the swaps. Notice we only save every 10th swap
  if(i%%10==0){
    rgbis[[c]]<-gbi_t
    c<-c+1
```

```
  }
}

#Here we convert our permuted GBIs to networks
rnets<-lapply(rgbis,get_network2)

#We can then calculate our reference distribution
ref_cvs<-unlist(lapply(rnets,CoV))

#We now are going to compare our Markov Chain with the observed coefficient of var
iation
#Unsurprisingly, our observed coefficient of variation lies outside the reference
 distribution, but then we knew our networks were non-random already
par(xpd=FALSE)
plot(ref_cvs,type="l",ylim=c(0,0.4),las=1,ylab="Value of test statistic",cex.lab=
1.5)
lines(x=c(-100,100000),y=c(obs_cv,obs_cv),col="red",lwd=2)
```



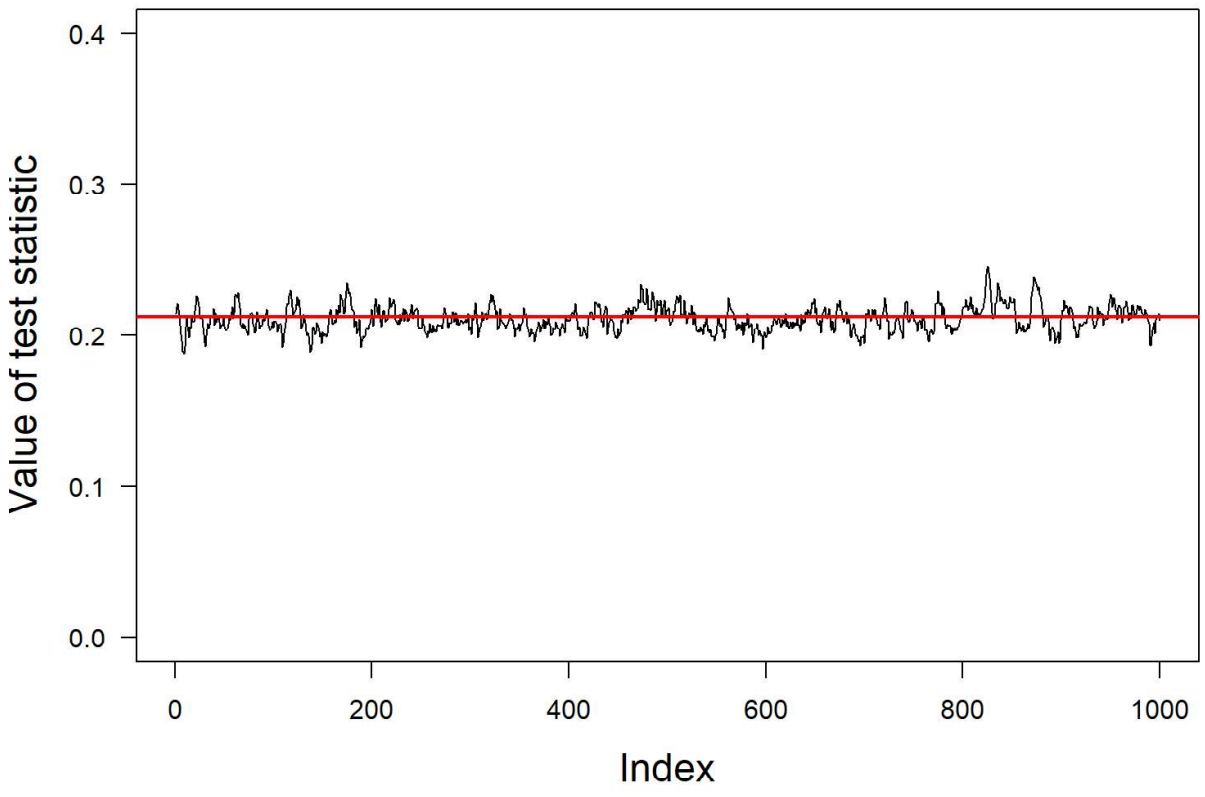

```
#Check p value from permutations
ref_cvs2<-c(obs_cv,ref_cvs)
sum(ref_cvs2<obs_cv)/length(ref_cvs2)
```

```
## [1] 0.998002
```

```
##############################################################################
```

```
#Therefore we generate a new reference model where we additionally constrain swaps
to be between individuals with the same nose colour (given that we have already es
tablished this to be important)

gbi_t<-gbi
rgbis<-list()

#Burn-in period
for(i in 1:1000){
  pind<-which(gbi_t>0,arr.ind=TRUE)
  tind1<-pind[sample(1:nrow(pind),1),]
  td<-which(day==day[tind1[1]])
  #This is where we work out the nose colour of the individual sampled first
  tn1<-noses[[1]][tind1[2]]
  pind2<-pind[which(pind[,1]%in%td),]
  tind2<-pind2[sample(1:nrow(pind2),1),]
  #This is where we work out the nose colour of the individual sampled second
  tn2<-noses[[1]][tind2[2]]
  if(tind1[1]!=tind2[1]&tind1[2]!=tind2[2]){
    if(gbi_t[tind1[1],tind2[2]]==0&gbi_t[tind2[1],tind1[2]]==0){
      #We only conduct a swap if they have the same nose colour. If not then the c
urrent permuted GBI is resampled in the Markov Chain
      if(tn1==tn2){
        gbi_t2<-gbi_t
        gbi_t2[tind2[1],tind1[2]]<-gbi_t[tind1[1],tind1[2]]
        gbi_t2[tind1[1],tind1[2]]<-gbi_t[tind2[1],tind1[2]]
        gbi_t2[tind1[1],tind2[2]]<-gbi_t[tind2[1],tind2[2]]
        gbi_t2[tind2[1],tind2[2]]<-gbi_t[tind1[1],tind2[2]]
        gbi_t<-gbi_t2
      }
    }
  }
}

#Sampling period
#100000 swaps with every 100th swap saved
c<-1
for(i in 1:100000){
  pind<-which(gbi_t>0,arr.ind=TRUE)
  tind1<-pind[sample(1:nrow(pind),1),]
  td<-which(day==day[tind1[1]])
  tn1<-noses[[1]][tind1[2]]
  pind2<-pind[which(pind[,1]%in%td),]
  tind2<-pind2[sample(1:nrow(pind2),1),]
  tn2<-noses[[1]][tind2[2]]
  if(tind1[1]!=tind2[1]&tind1[2]!=tind2[2]){
    if(gbi_t[tind1[1],tind2[2]]==0&gbi_t[tind2[1],tind1[2]]==0){
      if(tn1==tn2){
        gbi_t2<-gbi_t
        gbi_t2[tind2[1],tind1[2]]<-gbi_t[tind1[1],tind1[2]]
        gbi_t2[tind1[1],tind1[2]]<-gbi_t[tind2[1],tind1[2]]
        gbi_t2[tind1[1],tind2[2]]<-gbi_t[tind2[1],tind2[2]]
        gbi_t2[tind2[1],tind2[2]]<-gbi_t[tind1[1],tind2[2]]
        gbi_t<-gbi_t2
      }
    }
  }
```

```
  if(i%%100==0){
    rgbis[[c]]<-gbi_t
    c<-c+1
  }
}

#Here we convert our permuted GBIs to networks
rnets<-lapply(rgbis,get_network2)

#We can then calculate our reference distribution
ref_cvs<-unlist(lapply(rnets,CoV))

#If we produce the same plot as before, we can see that now the observed coefficie
nt of variation lies within the reference distribution, suggesting that individual
s interact at random aside from assorting by nose colour (this is the result we ex
pect from how we simulated the data)
plot(ref_cvs,type="l",ylim=c(0,0.4),ylab="Value of test statistic",las=1,cex.lab=
1.5)
lines(x=c(-100,100000),y=c(obs_cv,obs_cv),col="red",lwd=2)
```




```
#Check p value from permutations
ref_cvs2<-c(obs_cv,ref_cvs)
sum(ref_cvs2<obs_cv)/length(ref_cvs2)
```

```
## [1] 0.5994006
```

```
#We could also produce histograms as we have previously
```

An important caveat to using datastream permutations of association data is that choosing the right constraints on swaps can be very important and have major effects on results. Without using constraints then datastream permutations will swap edges at random and the reference distribution generated will be drawn from the configuration model. If it's unreasonable to expect relationships to be random within the constraints imposed then this can lead to errors in statistical inference.

Datastream permutations of are also not appropriate for testing the statistical significance of linear regressions (why we used node swaps) as they change the distribution of the response variable. This is discussed at length here: https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/2041-210X.13508

Section 3.1.4.2 – Permutations applied to edge lists

We can also use datastream permutations in edge lists of interaction data

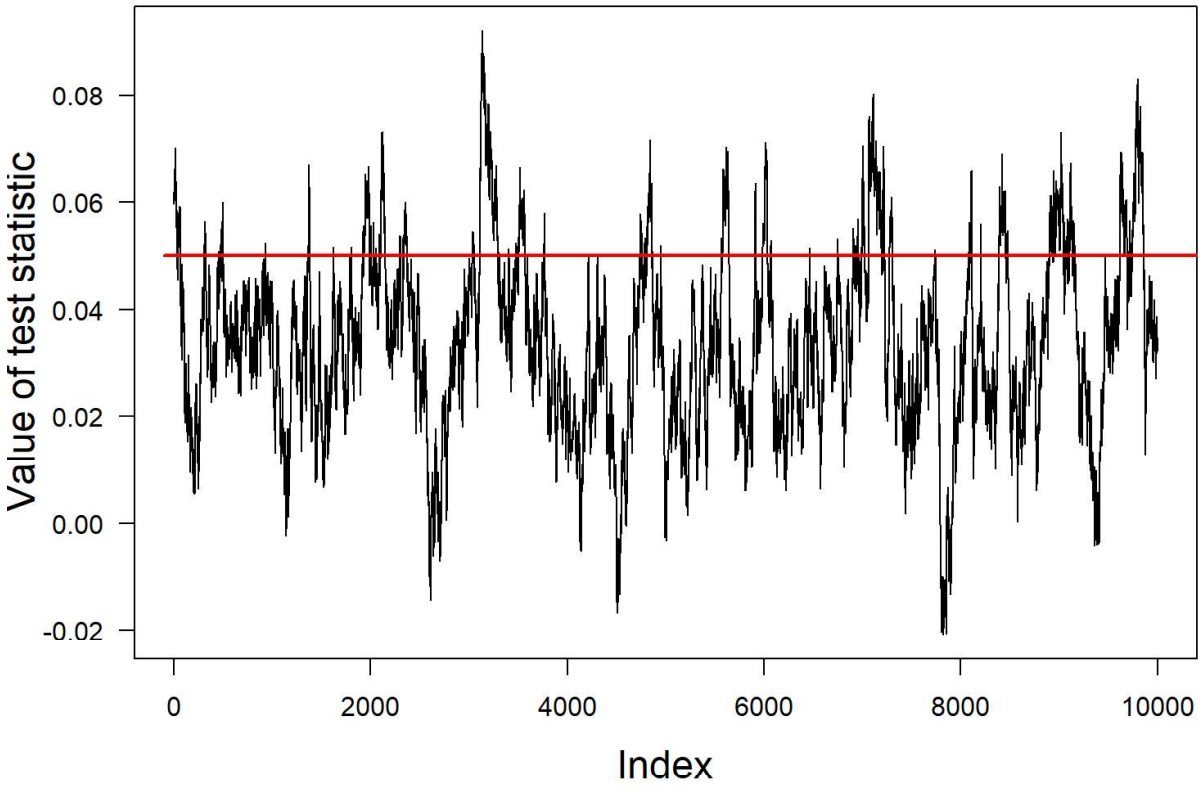
Before we were unable to test whether affiliative interactions were assorted by nose colour when you controlled for the structure of the association network. Using datastream permutations we can now test this hypothesis.

```
#Define function to convert permuted edge lists into adjacency matrices
matrix_gen<-function(a){
  Taff_net<-graph_from_edgelist(cbind(a,REC), directed = TRUE)
  E(Taff_net)$weight <- 1
  Taff_net<-simplify(Taff_net, edge.attr.comb=list(weight="sum"))
  b<-as_adjacency_matrix(Taff_net,sparse=FALSE,attr="weight")
  return(b)
}

#Define function to calculated assortativity in permuted networks
assortment2<-function(a){
  b<-assortnet::assortment.discrete(a, types=noses[[1]], weighted = TRUE, SE = FAL
SE, M = 1)
  return(b$r)
}

#We are going to swap the individual initiating affiliative interactions within su
b-groups to demonstrate that there are no more dominance interactions between indi
viduals of the same nose colour than expected by chance

#Calculate test statistic (assortativity) in observed network
obs<-assortnet::assortment.discrete(MAT_AFF, types=noses[[1]], weighted = TRUE, SE
= FALSE, M = 1)$r

#Generate reference distribution

T_W<-GIV
rGIV<-list()

#Burn-in period
for(i in 1:500){
  t1<-sample(1:length(T_W),1)
  tgr<-grA[t1]
  tw<-which(grA==tgr)
```

```
  t2<-sample(tw,1)
  T_W2<-T_W
  T_W2[t1]<-T_W[t2]
  T_W2[t2]<-T_W[t1]
  T_W<-T_W2
}

#Sampling period
c<-1
for(i in 1:100000){
  t1<-sample(1:length(T_W),1)
  tgr<-grA[t1]
  tw<-which(grA==tgr)
  t2<-sample(tw,1)
  T_W2<-T_W
  T_W2[t1]<-T_W[t2]
  T_W2[t2]<-T_W[t1]
  T_W<-T_W2
  if(i%%10==0){
    rGIV[[c]]<-T_W
    c<-c+1
  }
}

#convert permuted edgelists into adjacency matrices
r_affnets<-lapply(rGIV,matrix_gen)

#Calculate assortativity in permuted networks to generate reference distribution
refs<-unlist(lapply(r_affnets,assortment2))

#Add observed value to the reference distribution
refs2<-c(obs,refs)

#Calculate p value (0.025<p<0.975 indicates that the null hypothesis is accepted a
s expected)
sum(obs<refs)/length(refs2)
```

```
## [1] 0.1561844
```

```
#Plot the Markov Chain (could use histograms instead)
plot(refs,type="l",las=1,ylab="Value of test statistic",cex.lab=1.5)
lines(x=c(-100,100000),y=c(obs,obs),col="red",lwd=2)
```




```
#As expected affiliative interactions aren't assorted by nose colour once you cont
rol for the structure of the association network
```

Section 3.2 – RESAMPLING-BASED REFERENCE MODELS

Resampling-based reference models can be used to test a range of hypotheses that may overlap or differ from the type of hypotheses tested using permutation-based reference models.

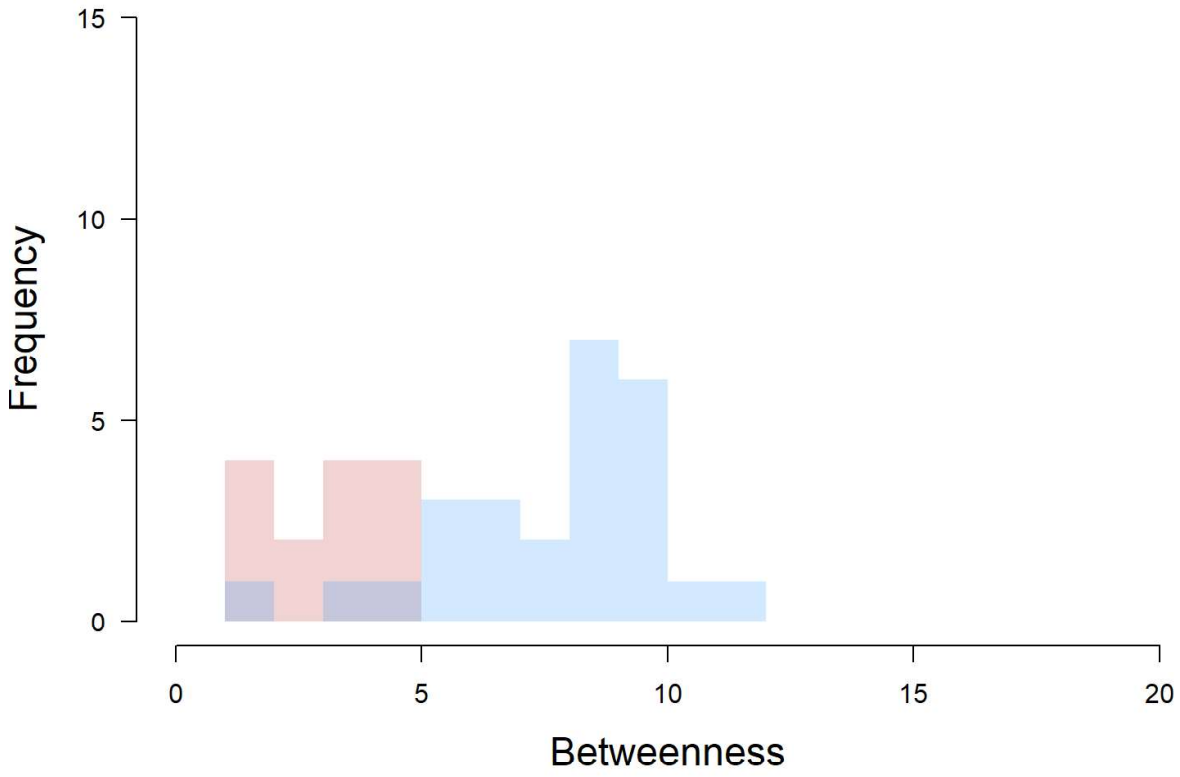
Resampling of either the network or the raw data can be used.

Section 3.2.1 – Network resampling

First we demonstrate how resampling of the network might be used. We show how it might be applied to test the hypothesis that two networks of different sizes are generated by the same underlying process. However, we provide two examples to demonstrate how challenging this can be. We also briefly discuss the relationship between network size and network properties.

Here we test whether the underlying structure of the huddling network in our smallest and largest groups are the same. We do this for winter when huddling networks are random graphs and summer when huddling networks are small-world graphs.

```
#Convert the winter huddling networks into adjacency matrices
hns2_m<-as_adjacency_matrix(hud_netSM_w,sparse=FALSE)
hnb2_m<-as_adjacency_matrix(hud_netBI_w,sparse=FALSE)

#Look at the degree distribution of the huddling network in the smallest (red) and
biggest (blue) groups
hist(colSums(hns2_m),col=adjustcolor("firebrick",0.2),border=NA,breaks=seq(0,20,1
),ylim=c(0,15),las=1,las=1,xlab="Betweenness",main="",cex.lab=1.5)
hist(colSums(hnb2_m),col=adjustcolor("dodgerblue",0.2),border=NA,breaks=seq(0,20,1
),add=TRUE)
```




```
#We then decide that the mean degree of the network is our test statistic. We calc
ulate the mean degree for each network
smeanw<-mean(colSums(hns2_m))
bmeanw<-mean(colSums(hnb2_m))

#Print mean degree of each network
print(smeanw)
```

```
## [1] 3.571429
```

```
print(bmeanw)
```

```
## [1] 8.153846
```

```
#Note that the mean degree of the huddling network in the biggest group is much la
rger

#We then generate our reference distribution by sampling from our larger network t
o produce a network equivalent in size to the smaller network and then recalculati
ng the mean degree
bmeansw<-numeric()
for(i in 1:1000){
  samp<-sample(1:nrow(hnb2_m),nrow(hns2_m),replace=FALSE)
  tm<-hnb2_m[samp,samp]
  bmeansw[i]<-mean(colSums(tm))
}

#Calculate the p value. Assuming alpha=0.05 then p<0.025 indicates the small netwo
rk has a larger mean degree than the sampled larger network and p>0.975 indicates
 it has a smaller degree
sum(smeanw<bmeansw)/length(bmeansw)
```

```
## [1] 0.878
```

```
#0.025<p<0.975 indicating the networks have similar mean degree, which is unsurpri
sing given they are generated by the same process

#An alternative comparison might instead to be to correct the degree measure by th
e number of individuals minus one (proportion of group connected with)
smeanC<-mean(colSums(hns2_m))/(nrow(hns2_m)-1)
bmeanC<-mean(colSums(hnb2_m))/(nrow(hnb2_m)-1)

#These values are now much more similar to each other. In both groups individuals
 are connected to about 30% of others
print(smeanC)
```

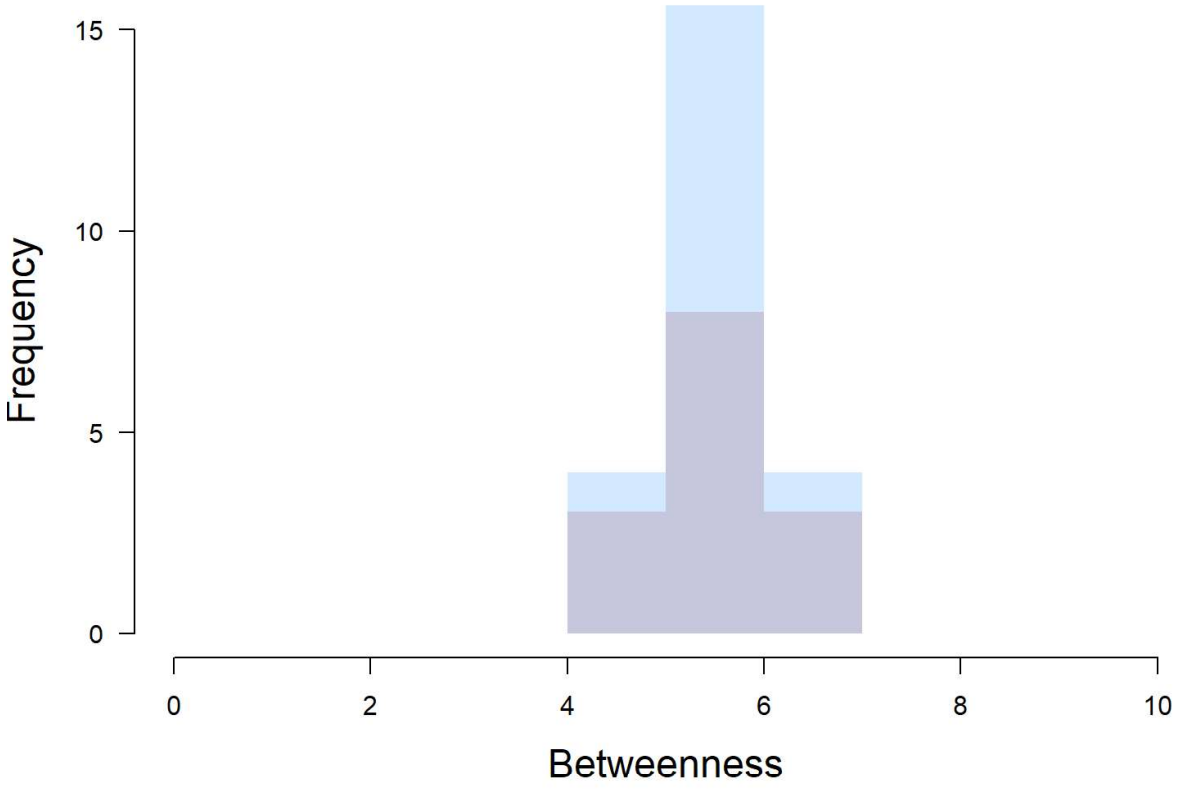
```
## [1] 0.2747253
```

```
print(bmeanC)
```

```
## [1] 0.3261538
```

```
##However, the success of using a process like this is dependent on the structure
 of the network (see main text) and so would need to be done with great care.

#When we do the same with the summer huddling networks, which have small-world pro
perties, then the degree distribution of the two networks is very similar even tho
ugh they are different sizes. When we do the same resampling procedure it indicate
s a difference between the small and large networks which we know are generated by
the same process.
hns_m<-as_adjacency_matrix(hud_netSM,sparse=FALSE)
hnb_m<-as_adjacency_matrix(hud_netBI,sparse=FALSE)

#Plot histogram comparing the degree distribution in the large and small groups
hist(colSums(hns_m),col=adjustcolor("firebrick",0.2),border=NA,breaks=seq(0,10,1),
ylim=c(0,15),las=1,xlab="Betweenness",main="",cex.lab=1.5)
```

```
hist(colSums(hnb_m),col=adjustcolor("dodgerblue",0.2),border=NA,breaks=seq(0,10,1
),add=TRUE)
```




```
#Calculate mean degree of each network as a test statistic
smean<-mean(colSums(hns_m))
bmean<-mean(colSums(hnb_m))

#As above we then generate our reference distribution by sampling from our larger
 network to produce a network equivalent in size to the smaller network and then r
ecalculating the mean degree
bmeans<-numeric()
for(i in 1:1000){
  samp<-sample(1:nrow(hnb_m),nrow(hns_m),replace=FALSE)
  tm<-hnb_m[samp,samp]
  bmeans[i]<-mean(colSums(tm))
}

#Again we could correct degree measure by the number of individuals minus one (pro
portion of group connected with)
smeanC2<-mean(colSums(hns_m))/(nrow(hns_m)-1)
bmeanC2<-mean(colSums(hnb_m))/(nrow(hnb_m)-1)

#This time the corrected values are very different from each other, similar to the
result detected by the resampling approach
print(smeanC2)
```

```
## [1] 0.4615385
```

```
print(bmeanC2)
```

```
## [1] 0.24
```

```
#If we looked at the uncorrected degrees then they are the same
print(mean(colSums(hns_m)))
```

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

```
print(mean(colSums(hns_m)))
```

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

```
#This is an example of where evaluation is important. If we think about the proces
ses structuring the two networks then we can see that the resampling approach and
 the test statistic used are not appropriate in this particular case. This is then
made clearer still when we use corrections for network size instead of resampling.
```

Comparing networks of different sizes is challenging (see main paper), but resampling can be useful in other contexts too. Below we provide an example of resampling the raw data used to construct the social networks.

In the main paper we provide an example of betweenness centrality calculated from our main study population and from study population B (which has similar social structure but is smaller). We also illustrate that briefly here.

Population A has higher “raw” betweenness values as the population is larger and therefore there are more potential paths between individuals. However, if we normalise the betweenness calculated by the number of potential dyads (pairs of nodes) in the network we see that (while more similar) the betweenness estimated in Population B is higher because there are fewer options of shortest paths between pairs of nodes in the smaller network.

```
#Calculate betweenness of the two population association networks
betA<-igraph::betweenness(full_net2,weights = 1/edge_attr(full_net2)$weight)
betB<-igraph::betweenness(full_net2_B,weights = 1/edge_attr(full_net2_B)$weight)

#Betweenness values are much higher (both mean and max) in the larger network
summary(betA)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0    42.0   511.9   371.0  7693.0
```

```
summary(betB)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0     0.0   106.7    59.5  1562.0
```

```
#However, when we normalise betweenness centrality values by the number of node pa
irs this difference disappears
betA2<-betA/(nrow(full_net)^2-nrow(full_net))
betB2<-betB/(nrow(full_net_B)^2-nrow(full_net_B))

#The distribution of betweenness values is now much more similar but normalised/co
rrected betweenness estimated in Population B is now higher as explained above
summary(betA2)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000000 0.0000000 0.0003822 0.0046578 0.0033762 0.0700051
```

```
summary(betB2)
```

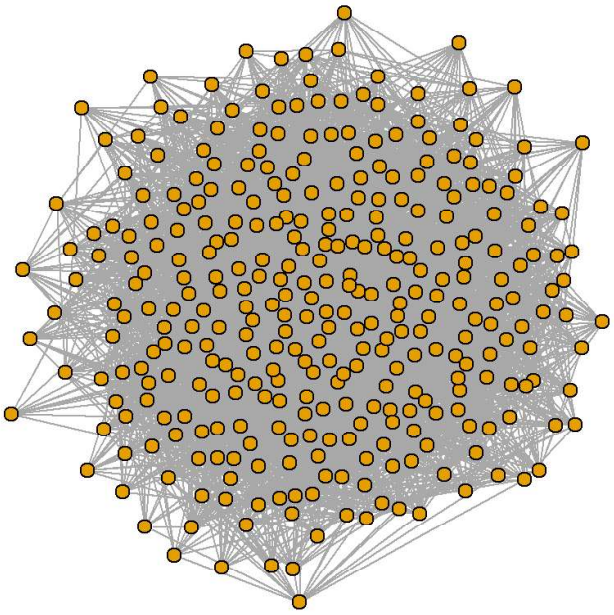
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.000000 0.000000 0.000000 0.009774 0.005449 0.143040
```

```
#Finding the correct approach to normalising measures is central to making appropr
iate comparisons between networks but also very challenging.
```

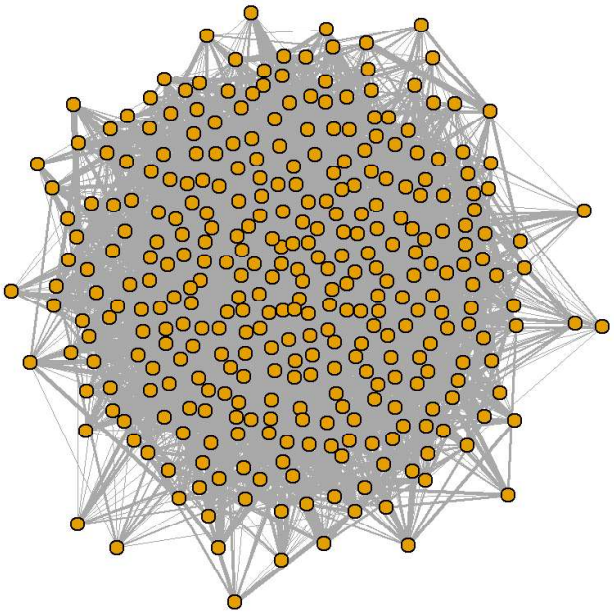
Another form of resampling-based reference model is to sample from a metric distribution, most commonly the degree distribution. For example, using igraph we can calculate the degree sequence of our full burbil association network and generate otherwise randomised graphs with the same degree sequence.

We can also resample edge weights from the observed network in a similar way.

```
#Calculate degree of the full population association network
deg<-igraph::degree(full_net2)

#Generate a reference network by resampling degree distribution
rdsn<-igraph::sample_degseq(deg,method="simple.no.multiple")

#Plot (randomised) reference network
plot(rdsn,vertex.label=NA,vertex.size=5)
```



```
#Note that this new network no longer has the discernible community structure of t
he observed association network

#We could then use resampling to reassign edge weights to this same graph (with or
without replacement). For example

#Resample edge weights with replacement
edge_attr(rdsn)$weight<-sample(edge_attr(full_net2)$weight,gsize(rdsn),replace=TRU
E)

#Plot rewired, weighted network
#You can see from these plots that we lose the strong grouping/community structure
of our original network without additional rules when we resample.
plot(rdsn,vertex.label=NA,vertex.size=5,edge.width=(edge_attr(rdsn)$weight*8)^2.5)
```



These types of reference model can be very useful when studying transmission through social networks. Degree distributions can have profound implications for spreading processes, and this type of approach provides a way to maintain the degree distribution to study the importance of other aspects of network structure.

Section 3.2.2 – Resampling the raw data

A more common (and frequently more useful) way to use resampling to generate a reference model is to resample the raw data itself.

We know that space use is important in structuring the overall (between-group) burbil association network. Therefore we could construct a reference model by resampling the spatial locations that subgroups were observed at and reconstructing the social network.

We use this example to demonstrate bootstrapping as a technique in generating reference models. This is sampling **with** replacement.

Here we test the hypothesis that there are no preferential associations between members of different groups when they meet.

N.B. Here we calculate only one instance of the reference distribution (a single reference network) but there is no reason this operation can't be repeated to generate a full reference distribution

```
#Generate adjacency matrix to store reference model
R_fn_rs<-matrix(0,nr=nrow(full_net),nc=ncol(full_net))

#We would probably make the assumption that these two probabilities (for within an
```

```
d between clan associations) were both 1 in the absence of any other information
R_p_wc<-1
R_p_bc<-1

#Resample within-group GBIs
R_gbis<-list()
for(i in 1:length(gbis)){
  nsg<-sample(1:nrow(gbis[[i]]),nrow(gbis[[i]]),replace=TRUE)
  R_gbis[[i]]<-gbis[[i]][nsg,]
}

#Resample between-group GBIs
R_sglocs<-list()
for(i in 1:length(sglocs)){
  nsg<-sample(1:nrow(sglocs[[i]]),nrow(sglocs[[i]]),replace=TRUE)
  R_sglocs[[i]]<-sglocs[[i]][nsg,]
}

#Recalculate new between-group associations using resampled spatial data
for(i in 1:100){
  for(j in 1:(n_groups-1)){
    for(k in (j+1):n_groups){
      tA<-paste0(R_sglocs[[j]][,1],"-",R_sglocs[[j]][,2])
      tB<-paste0(R_sglocs[[k]][,1],"-",R_sglocs[[k]][,2])
      tA2<-tA[daysl[[j]]==i]
      tB2<-tB[daysl[[k]]==i]
      tt<-match(tA2,tB2)
      if(sum(is.na(tt))<length(tt)){
      if(group_clans[j]==group_clans[k]){same<-rbinom(1,1,R_p_wc)}
      if(group_clans[j]!=group_clans[k]){same<-rbinom(1,1,R_p_bc)}
      if(same==1){
        paste(i,j,k)
        for(m in length(tt)){
          if(is.na(tt[m])==FALSE){
            tsg1<-which(tA==tA2[m]&daysl[[j]]==i)
            tsg2<-which(tB==tB2[tt[m]]&daysl[[k]]==i)
            tid1<-which(gbis[[j]][tsg1,]==1)
            tid2<-which(gbis[[k]][tsg2,]==1)
            tid1a<-inds_tot[g_tot==j&gi_tot%in%tid1]
            tid2a<-inds_tot[g_tot==k&gi_tot%in%tid2]
            R_fn_rs[tid1a,tid2a]<-R_fn_rs[tid1a,tid2a]+1
            R_fn_rs[tid2a,tid1a]<-R_fn_rs[tid1a,tid2a]
          }
        }
      }
      }
    }
  }
}

#Convert between-group associations to SRIs
for(i in 1:(nrow(full_net)-1)){
  for(j in (i+1):nrow(full_net)){
    R_fn_rs[i,j]<-R_fn_rs[i,j]/(200-full_net[i,j])
    R_fn_rs[j,i]<-R_fn_rs[i,j]
  }
}
```

```
#Add within-group associations to the population network
for(i in 1:n_groups){
  R_fn_rs[inds_tot[g_tot==i],inds_tot[g_tot==i]]<-get_network2(R_gbis[[i]])
}

#################################

#We now calculate our test statistics. We choose three different test statistics.
 We do this in a different way to how we have used our test statistic before; this
time our test statistic is a comparison to the observed dataset. We can use this t
o quantify how well different reference models do in recreating the observed data.

#The first test statistic is to calculate the correlation between the network gene
rated using resampled data and the observed association network (a Mantel test)
vegan::mantel(R_fn_rs,full_net)
```

```
##
## Mantel statistic based on Pearson's product-moment correlation
##
## Call:
## vegan::mantel(xdis = R_fn_rs, ydis = full_net)
##
## Mantel statistic r: 0.9828
##       Significance: 0.001
##
## Upper quantiles of permutations (null model):
##     90%     95%   97.5%     99%
## 0.00572 0.00742 0.00911 0.01085
## Permutation: free
## Number of permutations: 999
```

```
#The second test statistic is the summed difference in values between the referenc
e network and observed network, which can highlight any bias in the edge weights o
f the reference network.
sum(R_fn_rs-full_net)
```

```
## [1] 14.00534
```

```
#The third test statistic is the summed absolute difference in values between the
 reference network and observed network, which shows how similar the reference net
work is to the observed network (smaller value is a better fit).
sum(abs(R_fn_rs-full_net))
```

```
## [1] 170.1541
```

What we see here is that the network generated from resampled GBIs and spatial locations is very similar to the observed association network. However, we may want to evaluate why this is the case. The very strong community structure to the network means that the use of the matrix correlation for the overall network is always likely to find a very strong correlation. The other two test statistics also suggest reasonable similarity, and this is likely to be that by simply resampling the GBIs and spatial locations within the groups we don't break down the social community structure and this is the overriding process explaining

network structure. Therefore, because of the way we have constructed the reference model we have not learned much about our network.

We revisit this example in Section 3.4 below.

Section 3.3 – DISTRIBUTION-BASED REFERENCE MODELS

Instead of resampling from existing measures we can also fit statistical models to distributions of network measures and then re-generate networks accordingly.

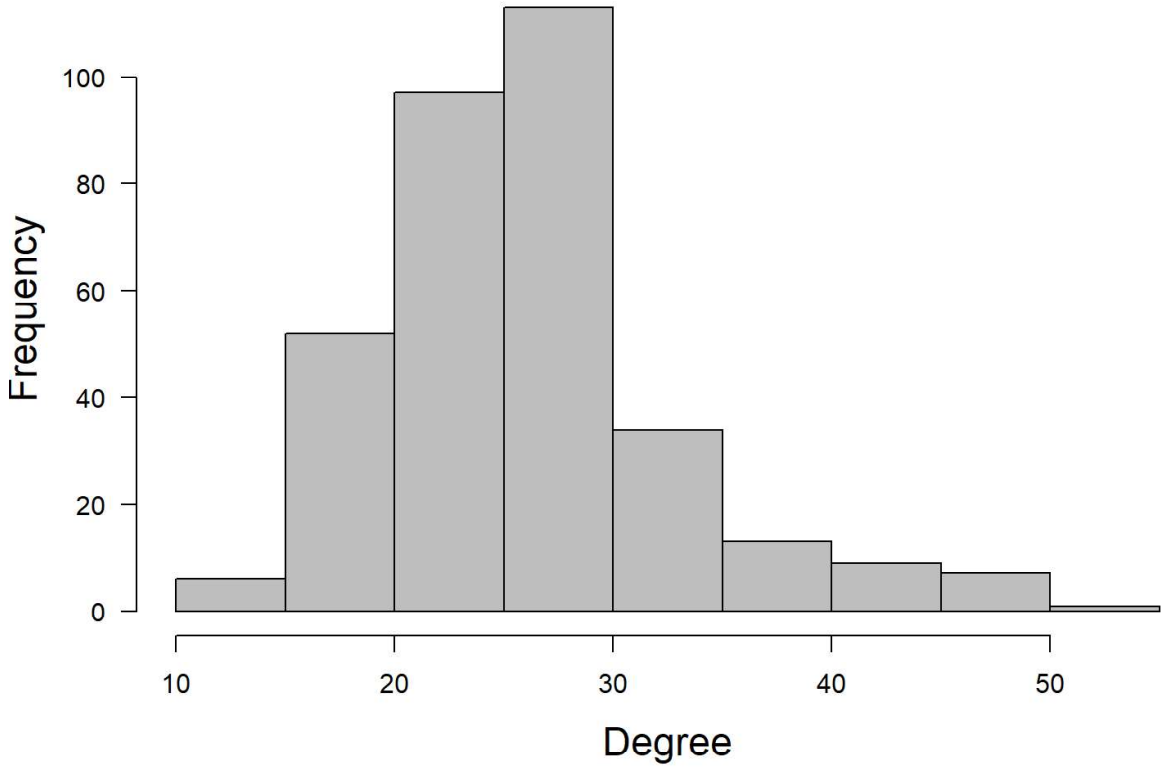
This can be a relatively easy process to follow (for some but not all network measures) when the distribution of only one measure is involved, but gets progressively more challenging if multiple properties of the network are to be retained. This is especially true when these distributions are correlated. We illustrate a burbil example in the main text, but also examine it briefly here.

Our example uses the overall population association network of our main burbil study population

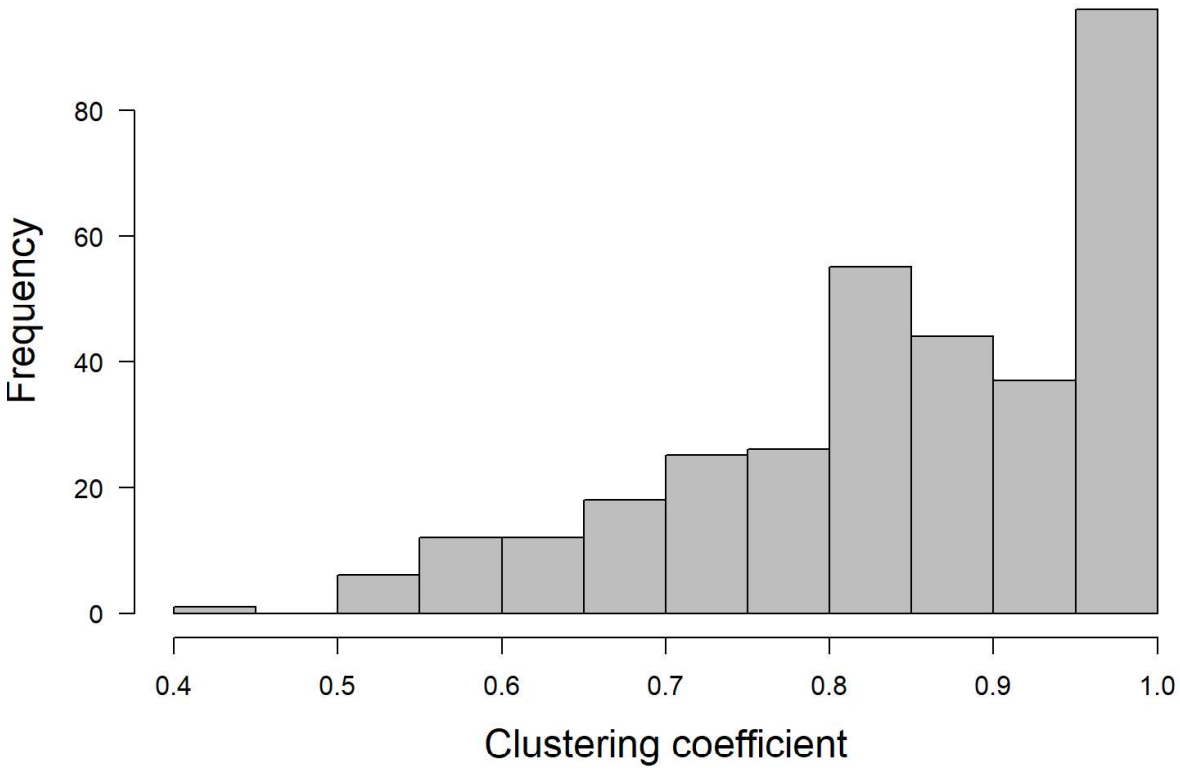
---

Please note that for this example, the version in the main paper is not completely identical to the version presented here although all of the code is identical and the general patterns are the same.

```
#We can calculate the degree distribution for our burbil study population as follo
ws
deg<-igraph::degree(full_net2)
#Plot histogram of the degree distribution
hist(deg,las=1,xlab="Degree",cex.lab=1.5,col="grey",main="")
```




```
#In a similar way we can also calculate the distribution of clustering coefficient
s for the network
clu<-igraph::transitivity(full_net2,type="weighted")
#Plot histogram of the distribution of clustering coefficients
hist(clu,las=1,xlab="Clustering coefficient",cex.lab=1.5,col="grey",main="")
```




```
#Our degree distribution is approximately Poisson distributed
mean(deg)
```

```
## [1] 26.42771
```

```
var(deg)
```

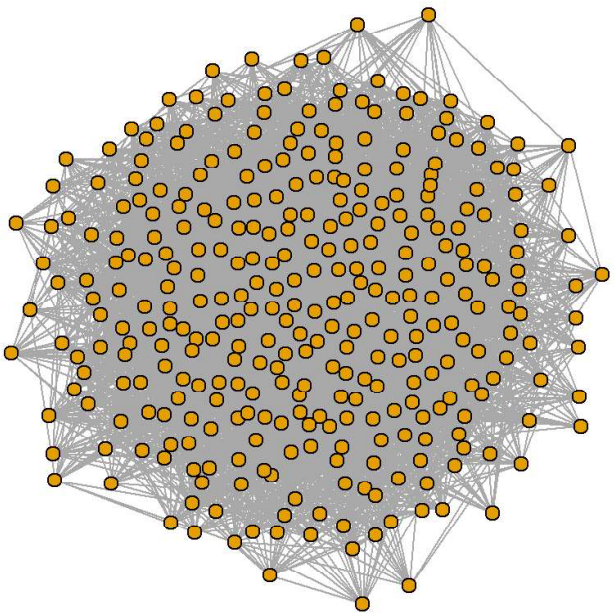
```
## [1] 44.90412
```

```
#The mean and variance are similar and degree is a count of the edges an individua
l has

#Therefore we can fit a Poisson distribution to our data (we could do the same wit
h other distributions such as the Negative Binomial if preferred)
fit<-glm(deg~1)

#We can then store the parameter for the Poisson distribution
mdeg<-coef(fit)

#Then using the sample_degseq() function introduced previously we can generate a n
etwork with the degree distribution drawn from that Poisson distribution

#N.B. Some proposed degree distributions generated by the Poisson distribution are
not realisable and so we have to use a While loop to keep trying until we generate
a suitable degree sequence.
```

```
ndeg<-rpois(length(V(full_net2)),mdeg)
while(class(try(igraph::sample_degseq(ndeg,method="simple.no.multiple")))=="try-error"){
  ndeg<-rpois(length(V(full_net2)),mdeg)
}
```

```
## Error in igraph::sample_degseq(ndeg, method = "simple.no.multiple") :
##   At games.c:937 : No simple undirected graph can realize the given degree sequence, Invalid value
## Error in igraph::sample_degseq(ndeg, method = "simple.no.multiple") :
##   At games.c:937 : No simple undirected graph can realize the given degree sequence, Invalid value
## Error in igraph::sample_degseq(ndeg, method = "simple.no.multiple") :
##   At games.c:937 : No simple undirected graph can realize the given degree sequence, Invalid value
```

```
rdsn2<-igraph::sample_degseq(ndeg,method="simple.no.multiple")

#Plot the reference network generated
plot(rdsn2,vertex.label=NA,vertex.size=5)
```

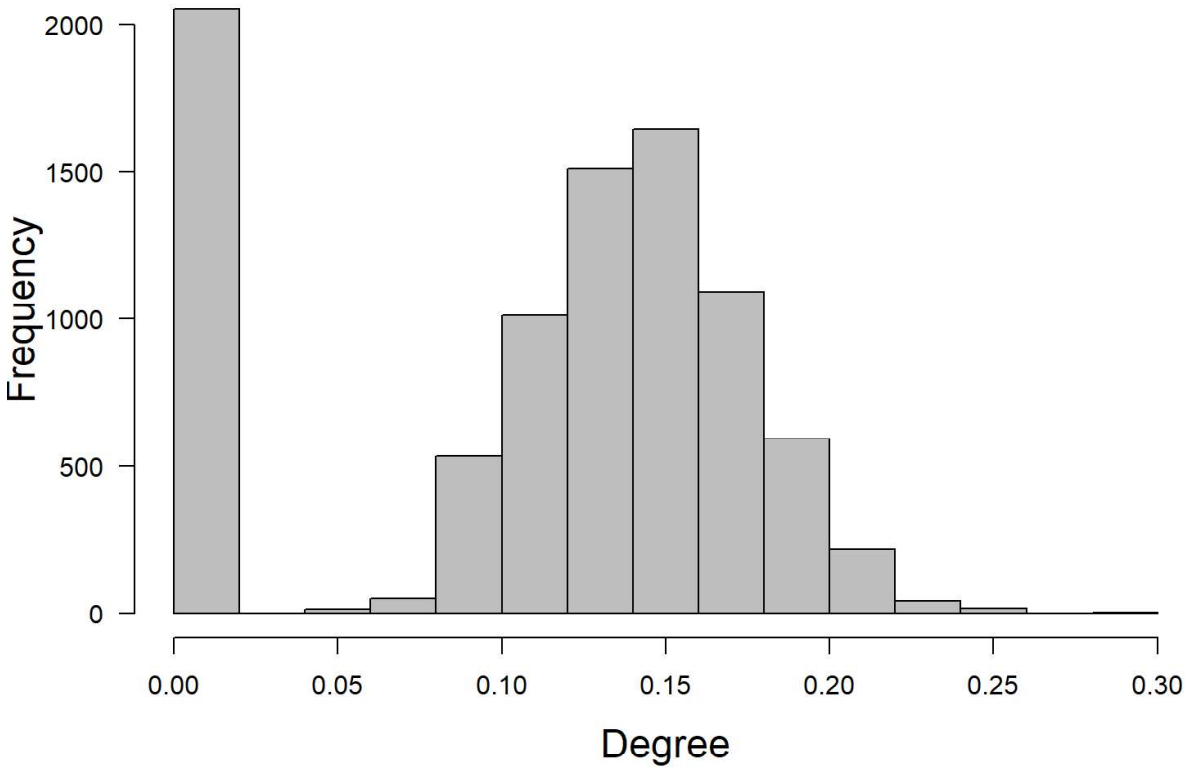


```
#Note that the network we generated here no longer has discernible community structure
```

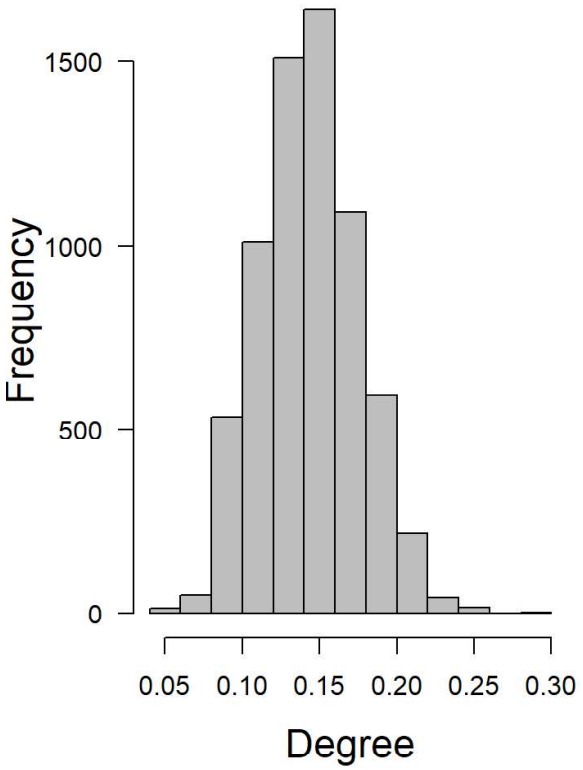
However, if we try to do the same with the distribution of clustering coefficients then we realise that it is not possible. Algorithms simply don't exist that would allow us to generate networks with a given distribution of many network measures. This is a key drawback of using distribution-based reference models.

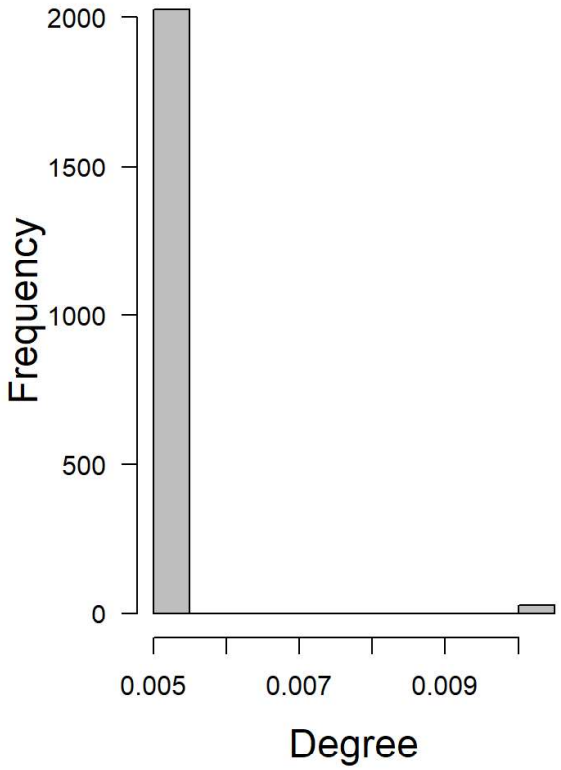
Another potential pitfall when using distribution-based (or even resampling-based) reference models is that it can be important to consider covariance between the values of different measures.

```
#As an example of this pitfall, we calculate the distribution for the edge weights
for our burbil study population
hist(full_net[full_net>0],las=1,xlab="Degree",cex.lab=1.5,col="grey",main="")
```



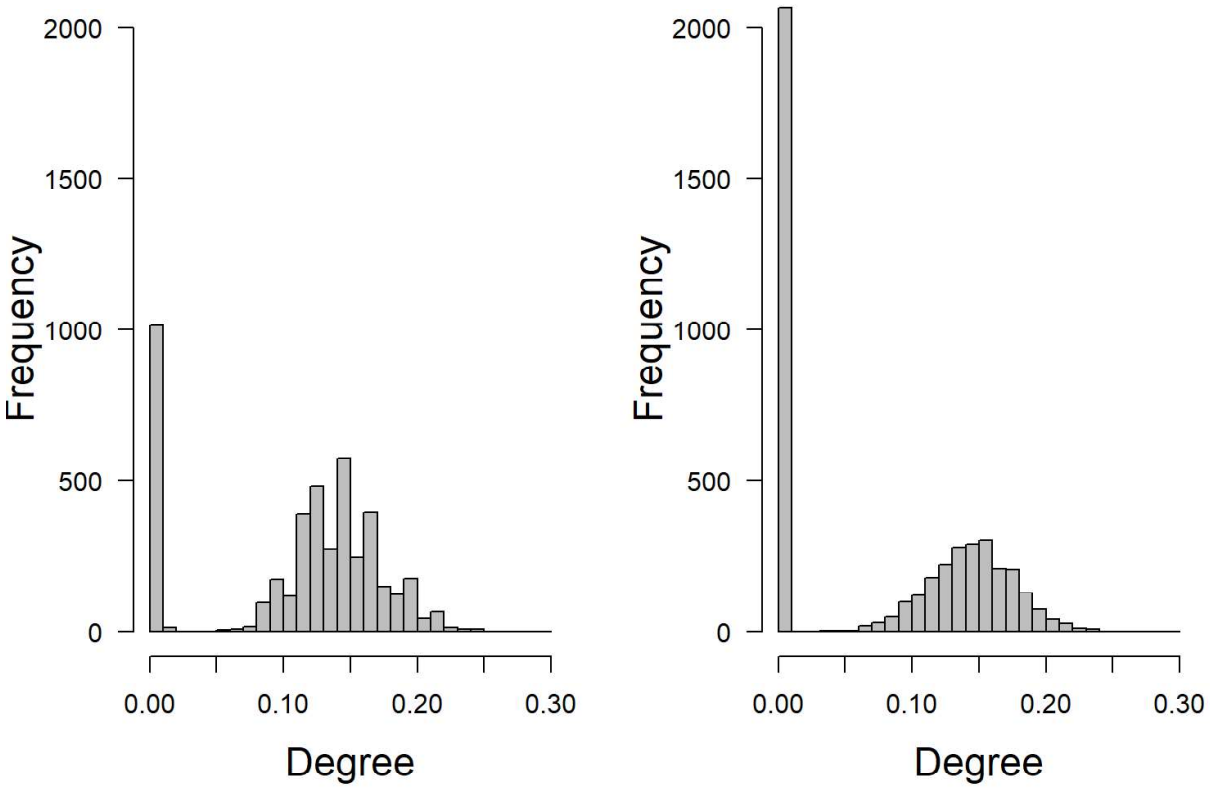

```
#This distribution is a little complex (it looks like there are two different stat
istical models generating it)

#We can examine these two processes by splitting the histogram. The two processes
 in this particular case (if you haven't worked it out) are caused in part by with
in-group versus between-group associations
par(mfrow=c(1,2))
hist(full_net[full_net>0.025],las=1,xlab="Degree",cex.lab=1.5,col="grey",main="")
hist(full_net[full_net>0&full_net<0.025],las=1,xlab="Degree",cex.lab=1.5,col="gre
y",main="")
```



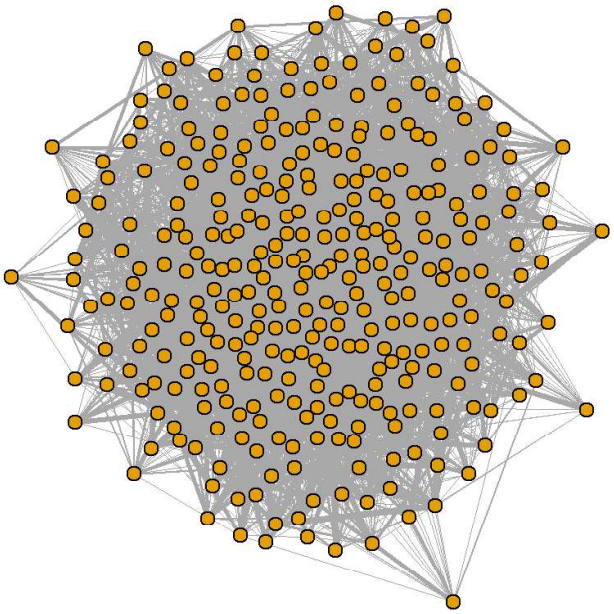



```
par(mfrow=c(1,1))

#We regenerate the distribution here in a slightly simplified form
#Calculate number of edges
ne<-gsize(full_net2)
#Calculate proportion of edges less than 0.025
pes<-sum(full_net>0&full_net<0.025)/ne

#mean and standard deviation of within-group edge weights
meb<-mean(full_net[full_net>0.025])
sdeb<-sd(full_net[full_net>0.025])

#set up vector to store edges for new graph
new_edgeweights<-rep(NA,gsize(rdsn2))

#fill in vector - we are effectively generating a normal distribution of within-gr
oup edge weights and making all between-group edge weights equal to 0.005
for(i in 1:gsize(rdsn2)){
  tb<-rbinom(1,1,pes)
  if(tb==1){
    new_edgeweights[i]<-0.005
  }
  if(tb==0){
    new_edgeweights[i]<-rnorm(1,meb,sdeb)
  }
}

#We can check our new edge weight degree distribution against the original one
```

```
par(mfrow=c(1,2))
hist(full_net[full_net>0&upper.tri(full_net)==TRUE],las=1,xlab="Degree",cex.lab=1.
5,col="grey",breaks=seq(0,0.3,0.01),main="",ylim=c(0,2000))
hist(new_edgeweights,las=1,xlab="Degree",cex.lab=1.5,col="grey",breaks=seq(0,0.3,
0.01),main="",ylim=c(0,2000))
```

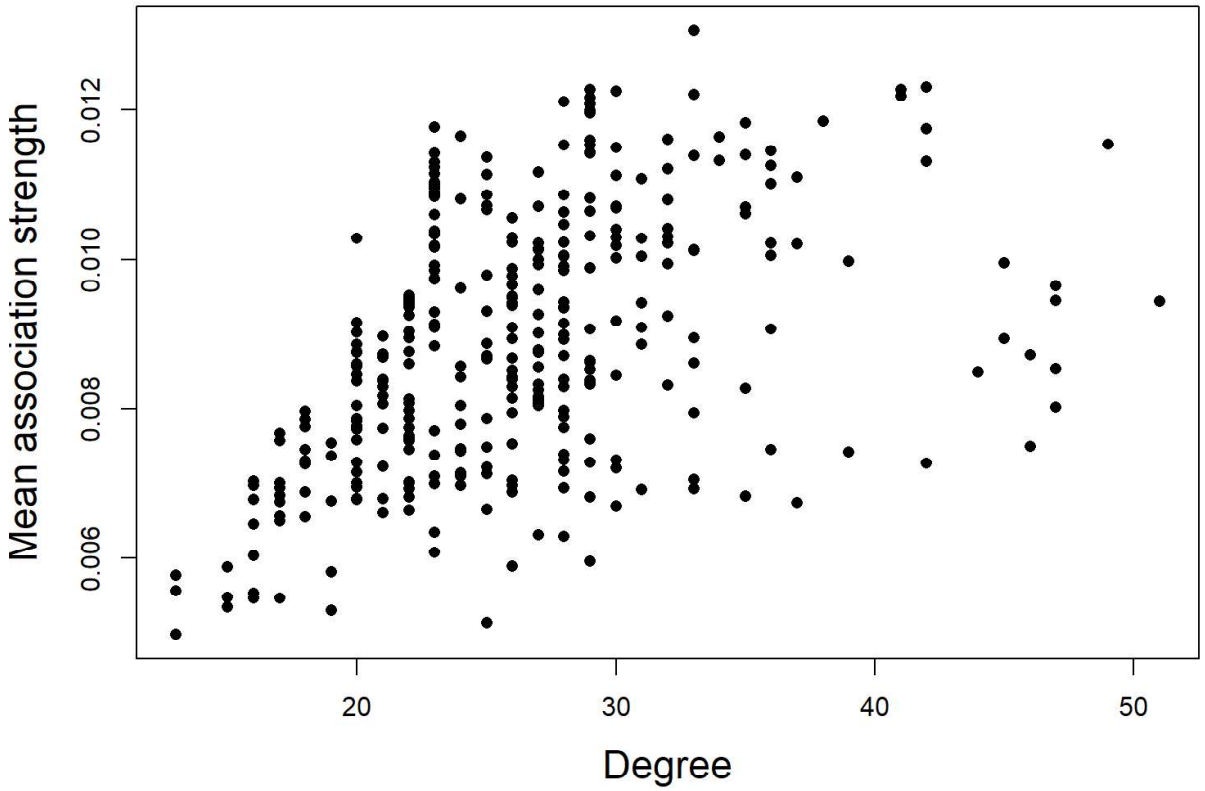



```
par(mfrow=c(1,1))

#We are feeling pretty pleased with ourselves, we have done a pretty good (albeit
 not perfect) job of fitting the edge weight distribution. The main weakness of ou
r current model is that it overstimates the number of very weak between-group conn
ections, which is unsurprising as we set all of these to the same very low value.
 We can use this for our new graph

#Plot newly generated, weighted reference network
edge_attr(rdsn2)$weight<-new_edgeweights
plot(rdsn2,vertex.label=NA,vertex.size=5,edge.width=(edge_attr(rdsn2)$weight*8)^3)
```



However, we then evaluate and realise there are problems with what we've done. Not only have we underestimated the strength of some between-group associations (presumably between members of nearby groups) but between-group edges tend to be much weaker and individuals with high degree may have more between-group connections reducing their mean edge weight. Similarly edge weights may also be biased by other things that influence associations such as nose colour. We failed to consider this covariance between edge weights and degree.

```
#So we plot the relationship here
plot(deg,rowMeans(full_net),pch=16,xlab="Degree",ylab="Mean association strength",
cex.lab=1.5)
```



From the plot we can work out that there is a complicated relationship between degree and the mean association strength and we need a complex reference model to capture this relationship properly.

Distribution-based reference models are complicated. For some network measures they might not be possible at all. For others it might be important to consider covariance between them in order to generate an appropriate reference model.

Distribution-based reference models could also be applied to raw data. You could fit a distribution to the relationship between individual traits or landscape features and the social or spatial data used to build your networks and rebuild your network from there. We don't cover that in this case study.

Section 3.4 – GENERATIVE REFERENCE MODELS

Our final type of reference model is the generative reference model. We first briefly illustrate the use of some basic statistical models for the networks themselves, before showing how agent-based models can be used to generate reference distributions of networks.

Section 3.4.1 – Statistical network models

Statistical network models are well-covered elsewhere in the network structure. We touch on two commonly used examples here:

- a. Stochastic block models which can be used to generate a reference distribution related to the community structure of the graph
- b. Exponential random graph models (ERGMs) which can be used to fit parameters to describe how the probability or weight of edges can be explained by structural properties of the network, nodal traits and dyadic traits.

```
#Note both these models are verbose during fitting and so we have hidden output an
d figures for this chunk of code

##Fit a stochastic block model to the association network
#We fit a block model for a weighted network, assuming edge weights have a Gaussia
n distribution as this is a reasonable assumption for our association network (see
previous sections)
sb<-blockmodels::BM_gaussian(membership_type="SBM_sym",adj=full_net,verbosity=0)
sb$estimate()
```

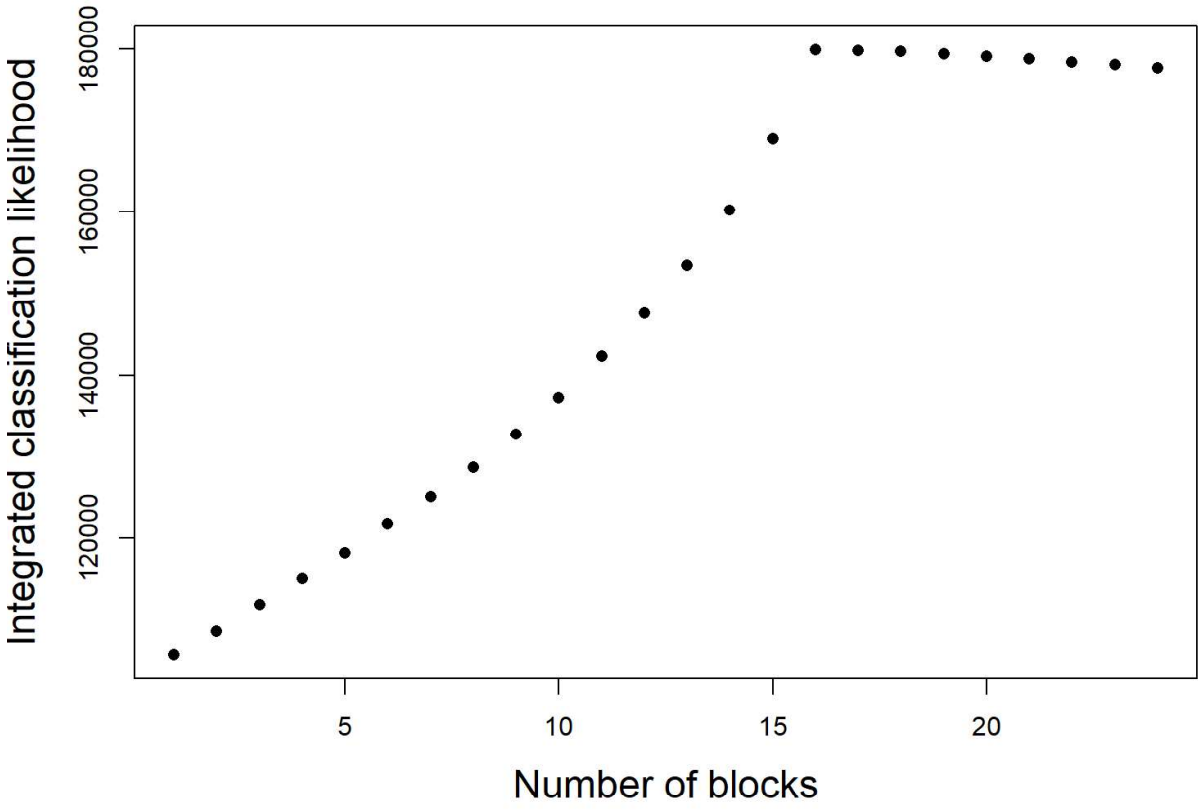
```
##Fit an ERGM to the dominance interaction data
#We first convert our dominance network to a network object for the ergm package i
n R
dom_el<-as.tnet(MAT_DOM)
```

```
## Warning in as.tnet(MAT_DOM): Data assumed to be weighted one-mode tnet (if this
## is not correct, specify type)
```

```
dom<-network(dom_el[,1:2])

#We then add edge weight as an attribute
network::set.edge.attribute(dom,"weight",as.vector(dom_el[,3]))

#We then add individual traits as node attributes
network::set.vertex.attribute(dom,"sex",as.vector(sexes[[1]]))
network::set.vertex.attribute(dom,"age",as.vector(ages[[1]]))
network::set.vertex.attribute(dom,"nose",as.vector(noses[[1]]))

#We can then fit a count ERGM (with a Poisson reference distribution) to the netwo
rk
#nonzero is a term to control for zero-inflation in edge counts (because many soci
al networks are sparse)
#Sum is an intercept-like term for edge weights
#We can then fit an array of terms to test hypotheses about the network structure
 and associations between connection weights and individual traits
#See https://rdrr.io/cran/ergm/man/ergm-terms.html for full details on ERGM terms

dom_mod<-ergm(dom~nonzero+sum+mutual(form="nabsdiff")+cyclicalweights(twopath="mi
n",combine="max",affect="min")+transitiveweights(twopath="min",combine="max",affec
t="min")+nodematch("sex",diff=TRUE)+nodematch("age",diff=TRUE)+nodematch("nose",di
ff=TRUE)+nodeofactor("age")+nodeofactor("sex")+nodeofactor("nose"),reference=~Pois
son,response="weight",silent=TRUE)

##To check that the model has converged we would run
#mcmc.diagnostics(dom_mod)
#(code not run here)
```

Now we can examine the fit of these statistical models and explore how to use them as reference distributions.

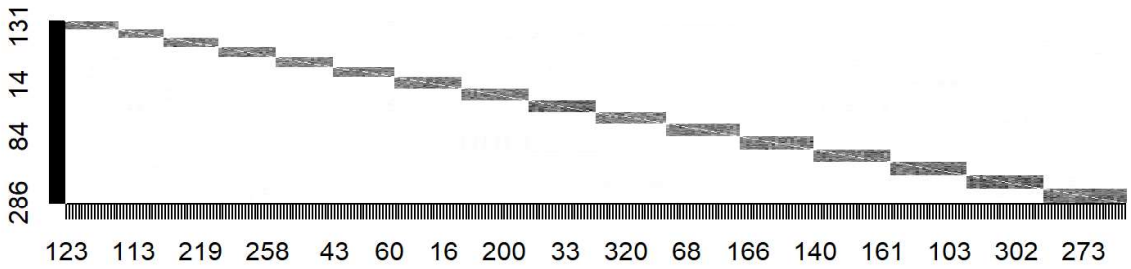
```
#For the stochastic block model, we can see how the fit of the model depends on th
e number of blocks or communities
plot(sb$ICL,pch=16,xlab="Number of blocks",ylab="Integrated classification likelih
ood",cex.lab=1.5)
```

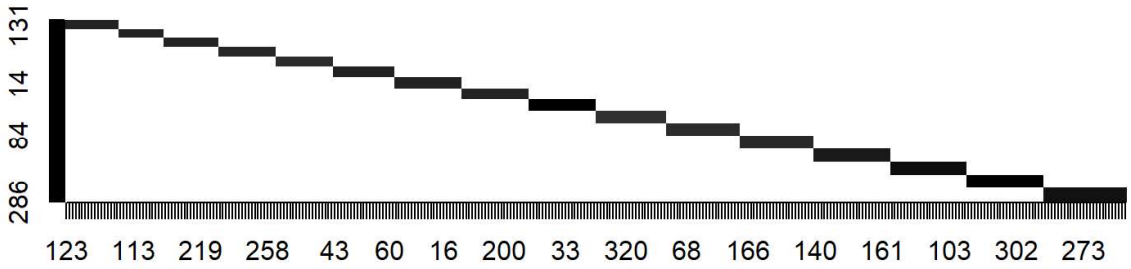



```
#We can see that the fit of the model doesn't really improve once 16 communities a
re included. This is unsurprising given we simulated 16 burbil groups and within-g
roup associations are so much more frequent than between-group associations.
#The best model fit is for 16 blocks/communities
which.max(sb$ICL)
```

```
## [1] 16
```

```
#We can examine the model predictions visually as follows
#The stochastic block model fits very closely to the observed network structure vi
sually
sb$plot_obs_pred(16)
```






```
#We can check the fit of the block model further by working out the memberships it
applies and comparing the size of blocks to the size of the groups we initially ge
nerated
mems<-sign(round(sb$memberships[[16]]$Z,2))
table(unlist(gss))
```

```
##
## 14 17 18 19 21 22 23 24 26
##  1  2  2  1  3  1  2  3  1
```

```
table(colSums(mems))
```

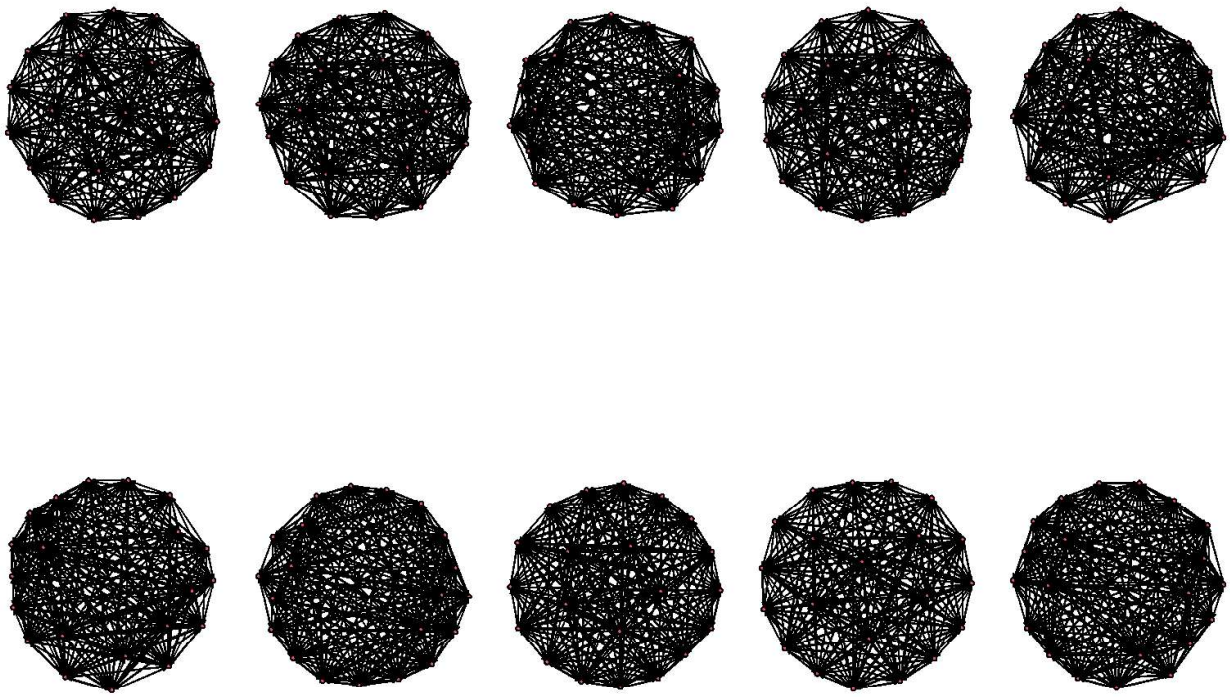
```
##
## 14 17 18 19 21 22 23 24 26
##  1  2  2  1  3  1  2  3  1
```

```
#We can see full model parameters using the command below (not run here)
#sb$model_parameters[16]

############################

#For the ERGM we can print out a model summary much like we do for other statistic
al models
summary(dom_mod)
```

```
## Call:
## ergm(formula = dom ~ nonzero + sum + mutual(form = "nabsdiff") +
##     cyclicalweights(twopath = "min", combine = "max", affect = "min") +
##     transitiveweights(twopath = "min", combine = "max", affect = "min") +
##     nodematch("sex", diff = TRUE) + nodematch("age", diff = TRUE) +
##     nodematch("nose", diff = TRUE) + nodeofactor("age") + nodeofactor("sex") +
##     nodeofactor("nose"), response = "weight", reference = ~Poisson,
##     silent = TRUE)
##
## Iterations:  5 out of 20
##
## Monte Carlo MLE Results:
##                                Estimate Std. Error MCMC % z value Pr(>|z|)
## nonzero                       -1.955482   0.409815      0  -4.772  < 1e-04 ***
## sum                            2.616254   0.059477      0  43.988  < 1e-04 ***
## mutual.nabsdiff               -0.213979   0.030494      0  -7.017  < 1e-04 ***
## cyclicalweights.min.max.min   -0.078708   0.021052      0  -3.739 0.000185 ***
## transitiveweights.min.max.min -0.014112   0.046224      0  -0.305 0.760138
## nodematch.sum.sex.F           -0.014817   0.049582      0  -0.299 0.765058
## nodematch.sum.sex.M            0.090035   0.043901      0   2.051 0.040280 *
## nodematch.sum.age.AD          -0.259056   0.055446      0  -4.672  < 1e-04 ***
## nodematch.sum.age.JUV          0.670305   0.073325      0   9.142  < 1e-04 ***
## nodematch.sum.age.SUB         -0.001308   0.077338      0  -0.017 0.986507
## nodematch.sum.nose.ORANGE      0.235533   0.063111      0   3.732 0.000190 ***
## nodematch.sum.nose.RED         0.185945   0.042246      0   4.402  < 1e-04 ***
## nodeofactor.sum.age.JUV       -0.851502   0.077000      0 -11.058  < 1e-04 ***
## nodeofactor.sum.age.SUB       -0.260616   0.043400      0  -6.005  < 1e-04 ***
## nodeofactor.sum.sex.M         -0.180688   0.042170      0  -4.285  < 1e-04 ***
## nodeofactor.sum.nose.RED      -0.045799   0.041396      0  -1.106 0.268567
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance:      0  on 420  degrees of freedom
##  Residual Deviance: -13286  on 404  degrees of freedom
##
## Note that the null model likelihood and deviance are defined to be 0.
## This means that all likelihood-based inference (LRT, Analysis of
## Deviance, AIC, BIC, etc.) is only valid between models with the same
## reference distribution and constraints.
##
## AIC: -13254    BIC: -13189    (Smaller is better.)
```

```
#We can also simulate networks based on the ERGM fit to provide a reference distri
bution for further hypothesis testing (for example, by seeing how goodness of fit
 changes for different regions of the network)

#Here we simulate 10 networks
ref_doms<-simulate(dom_mod,10)

#A quick plot to show the 10 reference networks
#N.B. we are plotting using the network package here for speed. We could convert t
o igraph if desired
par(mfrow=c(2,5),mar=c(0,0,0,0))
for(i in 1:length(ref_doms)){
```

```
  plot(ref_doms[[1]])
}
```



```
par(mfrow=c(1,1),mar=c(5,6,2,2))

#Here is the conversion into adjacency matrices
ref_mats<-as.sociomatrix(ref_doms,attrname="weight",simplify=FALSE)

#Print an adjacency matrix to demonstrate
ref_mats[[1]]
```

```
##      1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
## 1    0  6  5  3  5  5  5  6  3 12  2 14  1 14 15 13  2 12 22  3  4
## 2   10  0 10  9  8 17 10 10 11 14  7  2 11 17 10 23  7 15 12  9  9
## 3   22 10  0 11 15 20 14 16 13 23 12 20  6 18 16 18 21 14 15 25  9
## 4    5 12  6  0 10  7  7  8  6 12  7  6  5 19  3 10  4 12 13  8  1
## 5   11 14  2 27  0 19 16  8  5 14 10 11  7 13 11 21 22 21 12 19  5
## 6   12  9  7  4  8  0  8  8  8 14  3 11  5 17  8  3  8 10 11  8  9
## 7   14 13  6 16  7  8  0 12  6 21  6 12  4 15 16 11 23 23 19  9  6
## 8   21 17  3 16 17 12 10  0  9 18  6 22 12 16 15 17 16 22 19 12 10
## 9   30 21 11 20 18 19 10 15  0 17 13 25 19 11 28 16 11 19 24 20  8
## 10   5  5  3  2  2  3  3  4  0  0  2  5  3 10  9  9  4  6  7  5  3
## 11 13 19  6 13 14 18 19 12  4 30  0 13  8 13 19 17 14 18 26  9  7
## 12 10  6  6  7  6  7  1  4  4 13  0  0  2 11  9  0  3 15 13  4  6
## 13 10 14  4 19 15 20 14 11  3 19 16 14  0 19 10 15 31 23 15 14  7
## 14  4  5  2  5  1  2  0  7  4 17  4  4  4  0 10  7  3 12  9  2  4
## 15 24  2  4  6  6  5  7  2  6 11  2 21  0 16  0 15  7 15 14  0  4
```

```
## 16  6  6  6  5  5  7  5  4  3 15  5 12  5 18  3  0  3 16  6  4  3
## 17 10 20  5 13  6 13  8  6  7 21  9  9  5 14 14  9  0  7  7 12  5
## 18  3  4  0  2  9  2  5  2  6 15  4  5  2  5  6  9  7  0 10  0  4
## 19  5  4  1  4  5  2  4  3  3  9  0  6  7  8  7  9  8 16  0  3  4
## 20  8 12  9 15 15 10  8  7  5 13 11 12 10 20  5 18  7 10 18  0  8
## 21 16 15  6 27 18 10 12  9  6 23  7 14 20 11 17 18 27 14 19 13  0
```

Section 3.4.2 – Agent-based reference models

Agent-based models offer a powerful way to develop reference distributions that depend on behavioural rules rather than the structure of the observed network. You can program individuals to behave in a particular way and record their interactions and associations to generate a simulated network.

This is of course how we generated our burbil society in the first place. Therefore, in order to demonstrate the use of agent-based models we are going to reuse some of our previous code and encourage you to examine the consequences of changing key parts of it.

First we fit a spatially explicit agent-based model (ABM).

Second we fit a spatially explicit ABM applied at a subgroup level.

Third we develop a socially explicit agent-based ABM to see whether it is better able to explain burbil association patterns.

Note that we only produce one simulation of each agent-based model here. However, stochastic agent-based models such as this can also be used to build reference distributions of test statistics if run multiple times. Give it a go if you fancy!

In this example our question is: how are between-group association networks structured by space-use?

Our test statistics will be the correlation between the network generated using the ABM and the observed between-group network (a Mantel test), the summed difference in values between the reference network and observed network, which can highlight any bias in the edge weights of the reference network and the summed absolute difference in values between the reference network and observed network, which shows how similar the reference network is to the observed network (smaller value is a better fit).

Note that these are the same test statistics used in one of our resampling examples.

Note also that we have learned our lesson and are creating networks of summed associations between burbils from different groups rather than the entire network.

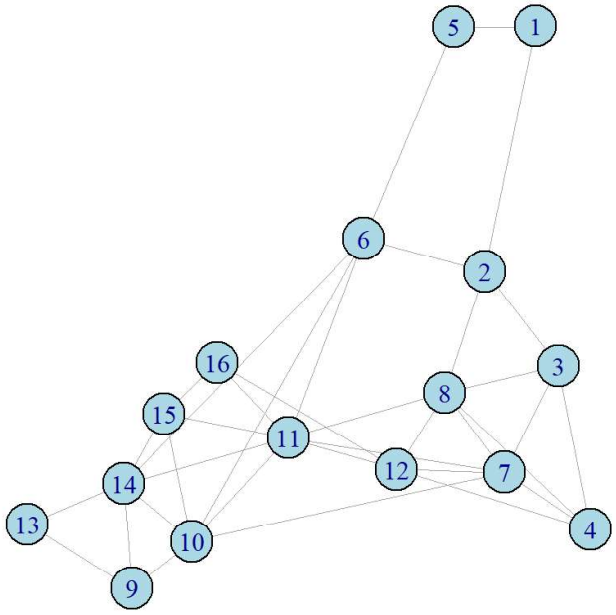
Finally note that we add parameters as we go along, e.g. dist_eff defined in the first code chunk is used in all three.

```
#Here we set the standard deviation for how far burbil subgroups tend to travel fr
om their home range centre (we will assume we know these for now)
#Note that we have used the value we originally used to generate the data here. Fe
el free to change the value and see what effect it has
#This will be used for all three reference models
dist_eff<-2
```

```
#First we need our group locations (printed below)
print(group_locs)
```

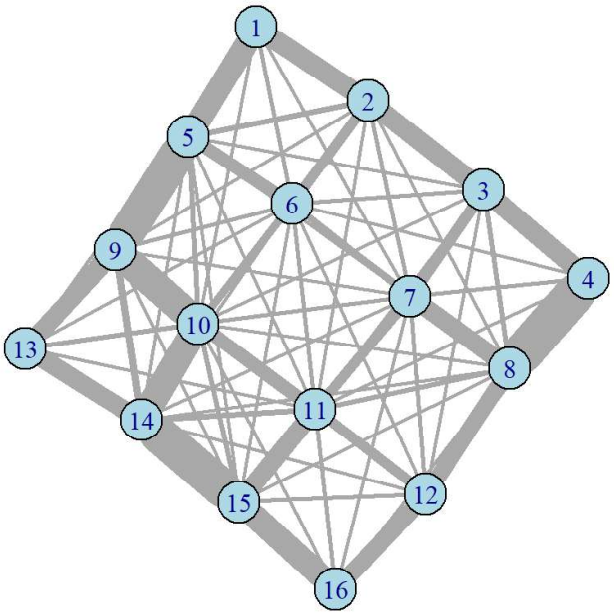
```
##      x  y
## 18   4  4
## 22   8  4
## 26  12  4
## 30  16  4
## 82   4  8
## 86   8  8
## 90  12  8
## 94  16  8
## 146  4 12
## 150  8 12
## 154 12 12
## 158 16 12
## 210  4 16
## 214  8 16
## 218 12 16
## 222 16 16
```

```
#We now create the observed between-group network
group_net<-matrix(0,nr=dim(group_locs)[1],nc=dim(group_locs)[1])
for(i in 1:nrow(full_net)){
  for(j in 1:ncol(full_net)){
    if(g_tot[i]!=g_tot[j]){
      group_net[g_tot[i],g_tot[j]]<-group_net[g_tot[i],g_tot[j]]+full_net[i,j]
    }
  }
}

#And we can then plot the observed between-group network
gnet<-graph.adjacency(group_net,mode="undirected",weighted=TRUE)
plot(gnet,vertex.color="light blue",edge.width=(edge_attr(gnet)$weight)^2)
```




```
############################################################


##We now generate our reference model with a truly spatially explicit ABM (i.e. we
remove the clan effect and allow individuals to be observed independently and not
 necessarily as subgroups)

#We assume that each individual is observed 100 times but this assumption can be c
hanged if desired

#Empty list to store new locations
R_indiv_locs<-list()

#Assign individual locations
for(i in 1:nrow(full_net)){
  tx<-round(rnorm(100,group_locs[g_tot[i],1],dist_eff))
  ty<-round(rnorm(100,group_locs[g_tot[i],2],dist_eff))
  R_indiv_locs[[i]]<-cbind(tx,ty)
}

#Generate full network for associations between individuals
R_fn<-matrix(NA,nr=nrow(full_net),nc=ncol(full_net))
for(i in 1:nrow(R_fn)){
  for(j in 1:ncol(R_fn)){
    R_fn[i,j]<-sum(rowSums(R_indiv_locs[[i]]==R_indiv_locs[[j]])==2)/100
  }
}
diag(R_fn)<-0
```

```
#Generate network of summed between-group associations
R_gn<-matrix(0,nr=dim(group_locs)[1],nc=dim(group_locs)[1])
for(i in 1:nrow(R_fn)){
  for(j in 1:ncol(R_fn)){
    if(g_tot[i]!=g_tot[j]){
      R_gn[g_tot[i],g_tot[j]]<-R_gn[g_tot[i],g_tot[j]]+R_fn[i,j]
    }
  }
}

#Plot network generated
RGN<-graph.adjacency(R_gn,mode="undirected",weighted=TRUE)
plot(RGN,vertex.color="light blue",edge.width=2+(edge_attr(RGN)$weight)^2)
```




```
#Calculate values for the test statistics
vegan::mantel(R_gn,group_net)
```

```
##
## Mantel statistic based on Pearson's product-moment correlation
##
## Call:
## vegan::mantel(xdis = R_gn, ydis = group_net)
##
## Mantel statistic r: 0.6505
##       Significance: 0.001
##
```

```
## Upper quantiles of permutations (null model):
##   90%   95% 97.5%   99%
## 0.135 0.176 0.211 0.251
## Permutation: free
## Number of permutations: 999
```

```
sum(R_gn-group_net)
```

```
## [1] 183.7163
```

```
sum(abs(R_gn-group_net))
```

```
## [1] 184.7219
```

Note for the first reference model, that while the network is fairly well correlated, the values of edge weights recorded are very different and upward biased.

The first reference model therefore does not explain our observed between-group network well at all. So we now go through and re-simulate subgroups (assuming we have knowledge about their typical properties) and assign a location to every subgroup instead of making the model purely individual-based.

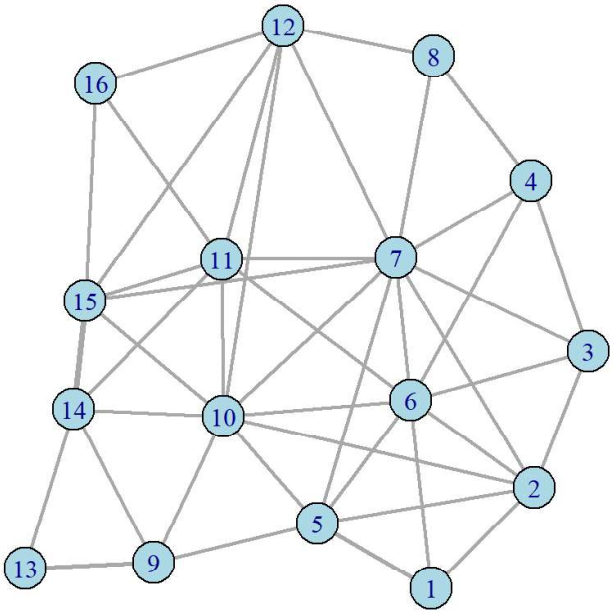
We maintain group sizes from the original population.

```
#We have copied/pasted code from where we first generated our GBIs and then change
d object names

#Create a list to store individual IDs
Rindss<-list()

#Create a list to store group sizes
Rgss<-list()

#Create a list to store the sex of each individual
Rsexes<-list()

#Create a list to store the age of each individual
Rages<-list()

#Create a list to store the nose colour of each individual
Rnoses<-list()

#Create a list to store information on which day a subgroup is observed on
Rdaysl<-list()

#Create a list to store a group-by-individual matrix for each burbil group
Rgbis<-list()

#Set the mean number of subgroups observed for each group each day
Rsg_mn<-5

#Set the strength of assortativity based on nose colour
#Set a number between 0 and 1
```

```
Rsg_ass<-0.2

#Generate association data within each burbil group!
for(j in 1:n_groups){

#individual identities
Rinds<-seq(1,n_inds[j],1)
Rindss[[j]]<-Rinds

#group size
gs<-length(Rinds)
Rgss[[j]]<-gs

#sex
sex<-sample(c("M","F"),gs,replace=TRUE)
Rsexes[[j]]<-sex

#age
age<-sample(c("AD","SUB","JUV"),gs,replace=TRUE,prob=c(0.6,0.2,0.2))
Rages[[j]]<-age

#nose
nose<-sample(c("RED","ORANGE"),gs,replace=TRUE,prob=c(0.7,0.3))
Rnoses[[j]]<-nose

################################

#Define number of subgroups on the first day
n_sg<-rpois(1,Rsg_mn-1)+1

#find halfway point
max_red<-floor(n_sg/2)

#Sample subgroups on the first day
subgroups1<-sample(n_sg,sum(nose=="RED"),replace=TRUE,prob=c(rep(0.5+Rsg_ass,max_r
ed),rep(0.5-Rsg_ass,n_sg-max_red)))
subgroups2<-sample(n_sg,sum(nose=="ORANGE"),replace=TRUE,prob=c(rep(0.5-Rsg_ass,ma
x_red),rep(0.5+Rsg_ass,n_sg-max_red)))

subgroups<-rep(NA,gs)
subgroups[nose=="RED"]<-subgroups1
subgroups[nose=="ORANGE"]<-subgroups2

#Store relevant information in the group-by-individual matrix and days vector
Rgbi<-matrix(0,nc=gs,nr=n_sg)
Rgbi[cbind(subgroups,seq(1,gs,1))]<-1
Rdays<-rep(1,nrow(Rgbi))

#Repeat process over 100 days of observations
for(i in 2:100){

  n_sg<-rpois(1,Rsg_mn-1)+1

  #find halfway point
  max_red<-floor(n_sg/2)

  subgroups1<-sample(n_sg,sum(nose=="RED"),replace=TRUE,prob=c(rep(0.5+Rsg_ass,max
```

```
_red),rep(0.5-Rsg_ass,n_sg-max_red)))
  subgroups2<-sample(n_sg,sum(nose=="ORANGE"),replace=TRUE,prob=c(rep(0.5-Rsg_ass,
max_red),rep(0.5+Rsg_ass,n_sg-max_red)))

  subgroups<-rep(NA,gss[[j]])
  subgroups[nose=="RED"]<-subgroups1
  subgroups[nose=="ORANGE"]<-subgroups2

  tgbi<-matrix(0,nc=gs,nr=n_sg)
  tgbi[cbind(subgroups,seq(1,gs,1))]<-1
  Rdays<-c(Rdays,rep(i,nrow(tgbi)))
  Rgbi<-rbind(Rgbi,tgbi)
}

#We edit the group-by-individual matrix and days vector to delete any "empty" grou
ps
Rgbi2<-Rgbi[rowSums(Rgbi)>0,]
Rdays<-Rdays[rowSums(Rgbi)>0]
Rgbi<-Rgbi2

Rdaysl[[j]]<-Rdays
Rgbis[[j]]<-Rgbi

}

Rsglocs<-list()
for(i in 1:n_groups){
  tx<-rep(NA,dim(Rgbis[[i]])[1])
  ty<-rep(NA,dim(Rgbis[[i]])[1])
  Rsglocs[[i]]<-data.frame(tx,ty)
  names(Rsglocs[[i]])<-c("x","y")
  Rsglocs[[i]]$x<-group_locs[i,1]+round(rnorm(dim(Rgbis[[i]])[1],0,dist_eff))
  Rsglocs[[i]]$y<-group_locs[i,2]+round(rnorm(dim(Rgbis[[i]])[1],0,dist_eff))
}

#We now calculate the full population association network
R_fn2<-matrix(0,nr=n_tot,nc=n_tot)

#Counts up between-group associations
for(i in 1:100){
  for(j in 1:(n_groups-1)){
    for(k in (j+1):n_groups){
      tA<-paste0(Rsglocs[[j]][,1],"-",Rsglocs[[j]][,2])
      tB<-paste0(Rsglocs[[k]][,1],"-",Rsglocs[[k]][,2])
      tA2<-tA[Rdaysl[[j]]==i]
      tB2<-tB[Rdaysl[[k]]==i]
      tt<-match(tA2,tB2)
      if(sum(is.na(tt))<length(tt)){
      #if(group_clans[j]==group_clans[k]){same<-rbinom(1,1,p_wc)} ###N.B.We have r
emoved clan effects
      #if(group_clans[j]!=group_clans[k]){same<-rbinom(1,1,p_bc)} ###N.B.We have r
emoved clan effects
      same<-1
      if(same==1){
        paste(i,j,k)
        for(m in length(tt)){
          if(is.na(tt[m])==FALSE){
```

```
            tsg1<-which(tA==tA2[m]&Rdaysl[[j]]==i)
            tsg2<-which(tB==tB2[tt[m]]&Rdaysl[[k]]==i)
            tid1<-which(Rgbis[[j]][tsg1,]==1)
            tid2<-which(Rgbis[[k]][tsg2,]==1)
            tid1a<-inds_tot[g_tot==j&gi_tot%in%tid1]
            tid2a<-inds_tot[g_tot==k&gi_tot%in%tid2]
            R_fn2[tid1a,tid2a]<-R_fn2[tid1a,tid2a]+1
            R_fn2[tid2a,tid1a]<-R_fn2[tid1a,tid2a]
          }
        }
      }
      }
    }
  }
}

#Create association network
for(i in 1:(nrow(R_fn2)-1)){
  for(j in (i+1):nrow(R_fn2)){
    R_fn2[i,j]<-R_fn2[i,j]/(200-R_fn2[i,j])
    R_fn2[j,i]<-R_fn2[i,j]
  }
}
for(i in 1:n_groups){
  R_fn2[inds_tot[g_tot==i],inds_tot[g_tot==i]]<-get_network2(Rgbis[[i]])
}

#Create between-group network
R_gn2<-matrix(0,nr=dim(group_locs)[1],nc=dim(group_locs)[1])
for(i in 1:nrow(R_fn2)){
  for(j in 1:ncol(R_fn2)){
    if(g_tot[i]!=g_tot[j]){
      R_gn2[g_tot[i],g_tot[j]]<-R_gn2[g_tot[i],g_tot[j]]+R_fn2[i,j]
    }
  }
}

#Plot between-group network from spatially explicit reference model with subgroups
RGN2<-graph.adjacency(R_gn2,mode="undirected",weighted=TRUE)
plot(RGN2,vertex.color="light blue",edge.width=2+(edge_attr(RGN2)$weight)^2)
```




```
#Calculate test statistics
vegan::mantel(R_gn2,group_net)
```

```
##
## Mantel statistic based on Pearson's product-moment correlation
##
## Call:
## vegan::mantel(xdis = R_gn2, ydis = group_net)
##
## Mantel statistic r: 0.5334
##       Significance: 0.001
##
## Upper quantiles of permutations (null model):
##   90%   95% 97.5%   99%
## 0.131 0.172 0.201 0.272
## Permutation: free
## Number of permutations: 999
```

```
sum(R_gn2-group_net)
```

```
## [1] 14.63693
```

```
sum(abs(R_gn2-group_net))
```

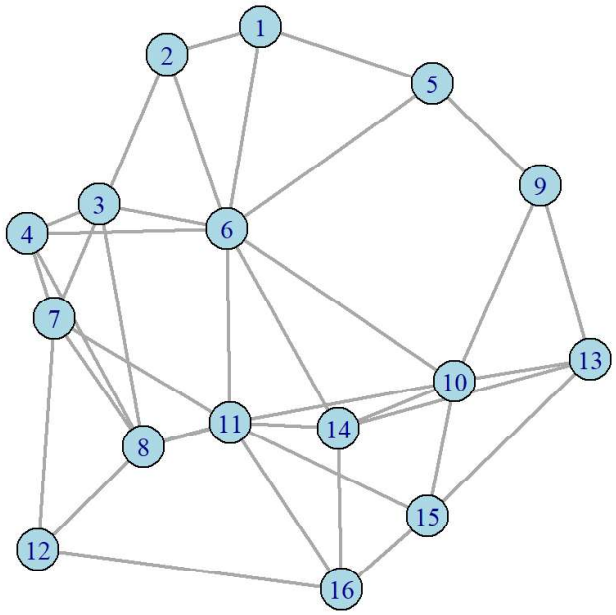
```
## [1] 19.96275
```

We can see from our test statistics that the correlation with the observed network is much poorer, but the edge weights are much more similar, although still seemingly overestimated on average.

We can now develop a third reference model that is socially-explicit, that is we included an effect of clan membership on whether between-group interactions occur between subgroups at the same location (you'll recall this is how we simulated our networks in the first place).

```
#We add our socially explicit parameters here. Note we have retained them as the o
riginal values used to create our burbil world. But please feel free to change the
m to see what effect it had on the network structure
Rp_wc<-p_wc
Rp_bc<-p_bc

#We now calculate the full population association network
R_fn3<-matrix(0,nr=n_tot,nc=n_tot)

#Counts up between-group associations
for(i in 1:100){
  for(j in 1:(n_groups-1)){
    for(k in (j+1):n_groups){
      tA<-paste0(Rsglocs[[j]][,1],"-",Rsglocs[[j]][,2])
      tB<-paste0(Rsglocs[[k]][,1],"-",Rsglocs[[k]][,2])
      tA2<-tA[daysl[[j]]==i]
      tB2<-tB[daysl[[k]]==i]
      tt<-match(tA2,tB2)
      if(sum(is.na(tt))<length(tt)){
      if(group_clans[j]==group_clans[k]){same<-rbinom(1,1,Rp_wc)}
      if(group_clans[j]!=group_clans[k]){same<-rbinom(1,1,Rp_bc)}
      #same<-1
      if(same==1){
        paste(i,j,k)
        for(m in length(tt)){
          if(is.na(tt[m])==FALSE){
            tsg1<-which(tA==tA2[m]&daysl[[j]]==i)
            tsg2<-which(tB==tB2[tt[m]]&daysl[[k]]==i)
            tid1<-which(Rgbis[[j]][tsg1,]==1)
            tid2<-which(Rgbis[[k]][tsg2,]==1)
            tid1a<-inds_tot[g_tot==j&gi_tot%in%tid1]
            tid2a<-inds_tot[g_tot==k&gi_tot%in%tid2]
            R_fn3[tid1a,tid2a]<-R_fn3[tid1a,tid2a]+1
            R_fn3[tid2a,tid1a]<-R_fn3[tid1a,tid2a]
          }
        }
      }
      }
    }
  }
}

#Create association network
for(i in 1:(nrow(R_fn3)-1)){
  for(j in (i+1):nrow(R_fn3)){
```

```
    R_fn3[i,j]<-R_fn3[i,j]/(200-R_fn3[i,j])
    R_fn3[j,i]<-R_fn3[i,j]
  }
}
for(i in 1:n_groups){
  R_fn3[inds_tot[g_tot==i],inds_tot[g_tot==i]]<-get_network2(gbis[[i]])
}

#Create between-group network
R_gn3<-matrix(0,nr=dim(group_locs)[1],nc=dim(group_locs)[1])
for(i in 1:nrow(R_fn3)){
  for(j in 1:ncol(R_fn3)){
    if(g_tot[i]!=g_tot[j]){
      R_gn3[g_tot[i],g_tot[j]]<-R_gn3[g_tot[i],g_tot[j]]+R_fn3[i,j]
    }
  }
}

#Plot between-group network from socially explicit reference model
RGN3<-graph.adjacency(R_gn3,mode="undirected",weighted=TRUE)
plot(RGN3,vertex.color="light blue",edge.width=2+(edge_attr(RGN3)$weight)^2)
```




```
#Calculate test statistics
vegan::mantel(R_gn3,group_net)
```

```
##
## Mantel statistic based on Pearson's product-moment correlation
```

```
##
## Call:
## vegan::mantel(xdis = R_gn3, ydis = group_net)
##
## Mantel statistic r: 0.283
##       Significance: 0.014
##
## Upper quantiles of permutations (null model):
##   90%   95% 97.5%   99%
## 0.112 0.160 0.200 0.297
## Permutation: free
## Number of permutations: 999
```

```
sum(R_gn3-group_net)
```

```
## [1] -1.616466
```

```
sum(abs(R_gn3-group_net))
```

```
## [1] 11.93412
```

The third reference model does a much better job of explaining the observed burbil association network, indicating that including the clan membership is an important factor driving between-group network structure.

When you use an agent-based model then you may want to pick specific values of key parameters and generate distributions of test statistics (as we have done here). However, you can also use values of your test statistic to fit agent-based models to your observed network using your chosen test statistic. For example, you could use a Markov Chain or even approximate Bayesian computation (ABC) to produce estimates for parameter values that generate networks most similar to the observed network according to the test statistic you have selected.

END