R Script for Archaeological Networks, Community Detection, and Critical Scales of Interaction in the U.S. Southwest/Mexican Northwest

Matthew A. Peeples and Robert J. Bischoff

2023-01-28

Install packages if necessary

```
packages = c(
  'sqldf',
  'compiler',
  'reshape',
  'msm',
  'snowfall',
  'parallel',
  'foreach',
  'doParallel'
  )
```

Uniform Probability Density Function (UPDF)

Load libraries and set up parallel processing.

```
require(sqldf)
require(compiler)
require(reshape)
require(msm)
require(snowfall)
require(parallel)
require(sqldf)

# Multicore setup
library(foreach)
library(doParallel)
cl <- makeCluster(detectCores())
registerDoParallel(cl)
```

Create functions

Create UPDF function. Note that the documented GitHub version of this function is available here: https://github.com/mpeeples2008/UniformProbabilityDensityAnalysis

```
updf <-
  function(site,
           cer.type,
           ct,
           start,
           end,
           chron,
           interval = 5,
           cutoff = 0.1,
           min.period = 25) {
    ## Remove any type with a count of 0 from all vectors
    trim <- which(ct > 0)
    cer.type <- as.vector(cer.type[trim])
    site <- as.vector(site[1])
    start <- start[trim]
    end <- end[trim]
    ct <- ct[trim]
    chron <- chron[trim]


    ## define function for rounding by base defined in "interval"
    ## argument and round start and end dates for each type
    mround <- function(x, base) {
      base * round(x / base)
    }
    start <- mround(start, interval)
    end <- mround(end, interval)

    ## find minimal ceramic intervals based on rounded type date overlaps
    years <- unique(sort(c(start, end)))
    period <-
      do.call(rbind, foreach(i = 1:(length(years) - 1)) %dopar%
                rbind(c(years[i], years[i + 1])))

    # Ensure that no period is shorter than the specificed min.period argument
    # and combine periods to satisfy this requirement
    if (length(period) > 2) {
      per.short <-
        foreach(i = 1:(nrow(period) - 1)) %dopar% (period[i + 1, 1] - period[i, 1])
      comb.per <- which(per.short < min.period)
      if (length(comb.per) > 0) {
        period <- period[-comb.per, ]
        for (i in 1:(nrow(period) - 1)) {
          period[i, 2] <- period[i + 1, 1]
        }
      }
    }

    ## Define lists of ceramic period lengths and site period lengths
    cer.per <-
      foreach(i = 1:length(start)) %dopar% (seq((start[i]:end[i])) +
                                              (start[i] - 1))
    per.len <-
```

```
  foreach(i = 1:nrow(period)) %dopar% (seq(period[i, 1]:period[i, 2]) +
                                         (period[i, 1]) - 1)

## Create Uniform Distance dataframe via parallel function
ucalc <- function(a, b) {
  out <- (length(intersect(a, b)) - 1) / (length(a) - 1)
  if (out < 0) {
    out <- 0
  }
  return(out)
}

udist <-
  foreach(a = cer.per, .combine = 'rbind') %:%
  foreach(b = per.len, .combine = 'c') %dopar% (ucalc(a, b))
if (length(udist) == 1) {
  udist <- as.matrix(udist)
}
colnames(udist) <- paste('AD', period[, 1], '-', period[, 2], sep = '')

## Create dataframe of prior values and sum to calculate prior probabilities
prior.tab <- udist * ct
if (ncol(prior.tab) > 1) {
  if (length(prior.tab[which(chron == 1), ]) == ncol(prior.tab)) {
    prior <-
      prior.tab[which(chron == 1), ] / sum(prior.tab[which(chron == 1), ])
  } else
  {
    prior <-
      colSums(prior.tab[which(chron == 1), ]) /
      sum(prior.tab[which(chron == 1), ])
  }
} else {
  prior <- udist[1, ]
}
prior[is.na(prior)] <- 0

## Create dataframe of count probabilities
pij <- sweep(prior.tab, 2, colSums(prior.tab), '/')
pij[is.na(pij)] <- 0

## Create dataframe of probabilities based on uniform distribution
uij <- sweep(udist, 2, colSums(udist), '/')
uij[is.na(uij)] <- 0

## Create dataframe of standard deviations of
## uniform distribution probabilities
sd.t <- sqrt(sweep((uij * (1 - uij)), 2, colSums(ceiling(udist)), '/'))

## Create dataframe of conditionals and calculate conditional probabilities
cij <- dnorm(pij, uij, sd.t)
cij[which(uij - sd.t == 1)] <-
  dnorm(1, 1, 1)
```

```
is.na(cij) <- !is.finite(cij)

# calculate conditional probability and rescale from 0-1
conditional <- apply(cij, 2, mean, na.rm = T)
conditional[is.na(conditional)] <- 0
if (sum(conditional) > 0) {
  conditional <- conditional / sum(conditional)
}

## Calculate posterior proportions and remove any generated NA
posterior <-
  foreach(i = 1:length(prior), .combine = 'c') %dopar%
  ((prior[i] * conditional[i]))
posterior <- posterior / sum(posterior)
posterior[is.na(posterior)] <- 0

## Deal with edge cases where sum of posterior
## probabilities = 0 or conditional = prior
if (sum(posterior) == 0) {
  posterior <- prior
}
if (identical(conditional, prior)) {
  posterior <- prior
}

## Create dataframe with ceramics apportioned by uniform distribution
## (udist.out) and posterior estimates (post.out)
cer.lab <-
  as.data.frame(cbind(rep(site, length(ct)), cer.type,
                      ct, start, end, chron))
colnames(cer.lab) <-
  c('Site', 'Type', 'Count', 'StartDate', 'EndDate', 'Chronology')

udist.out <- cbind(cer.lab, (udist * ct))

post.tab <-
  (((sweep(
    prior.tab, 2, posterior, '*'
  )) / rowSums(sweep(
    prior.tab, 2, posterior, '*'
  ))) * ct)
post.tab[is.na(post.tab)] <- 0
colnames(post.tab) <- paste('AD', period[, 1], '-', period[, 2], sep =
                            '')
post.out <- cbind(cer.lab, post.tab)

# calculated beginning (lwr) and ending (upr) dates based on the first
# and last period with posterior probability about the selected threshold
lwr <- period[min(which(posterior > cutoff * max(posterior))), 1]
upr <- period[max(which(posterior > cutoff * max(posterior))), 2]

# Reduced ceramic list for occupation period as selected by cutoff
extra <-
```

```
    as.matrix(post.tab[, -c((min(which(
      posterior > cutoff * max(posterior)
    ))):max(which(posterior > cutoff * max(posterior))))])
  colnames(extra) <-
    colnames(post.tab)[-c((min(which(
      posterior > cutoff * max(posterior)
    ))):max(which(posterior > cutoff * max(posterior))))]
  occ.cer <-
    as.matrix(post.tab[, c((min(which(
      posterior > cutoff * max(posterior)
    ))):max(which(posterior > cutoff * max(posterior))))])
  colnames(occ.cer) <-
    colnames(post.tab)[c((min(which(
      posterior > cutoff * max(posterior)
    ))):max(which(posterior > cutoff * max(posterior))))]

  if (length(extra) > 0) {
    occ.cer.out <-
      occ.cer / rowSums(occ.cer) * (rowSums(extra)) + occ.cer
  } else {
    occ.cer.out <- occ.cer
  }

  occ.cer.out[is.na(occ.cer.out)] <- 0

  occ.cer.out <- cbind(cer.lab, occ.cer.out)

  per.occ <-
    period[c((min(which(
      posterior > cutoff * max(posterior)
    ))):max(which(posterior > cutoff * max(posterior)))), ]

  ## Create output list and return
  out.list <-
    list(
      site = site,
      prior = prior,
      posterior = posterior,
      conditional = conditional,
      period = period,
      samp.size = sum(ct),
      udist = udist.out,
      postdist = post.out,
      occ = cbind(lwr, upr),
      per.occ = per.occ,
      cer.occ = occ.cer.out
    )
  return(out.list)
}
```

Read in ceramic data in long format

```
dat.long <- read.csv('data_all.csv',header=T)
sites <- unique(dat.long$Site)
```

Run UPDA for all sites and compile ceramic info by 25 year interval

```
out.cer <- list()
beg.per <- 300
end.per <- 2000
interval <- 25
full.per <- seq(beg.per,end.per,by=interval)
full.lab <- do.call(rbind,foreach(i=1:(length(full.per)-1)) %dopar%
                      rbind(c(full.per[i],full.per[i+1])))


for (i in 1:length(sites)) {
  qv <- dat.long[which(dat.long$Site==sites[i]),]
  vv <- qv[which(qv$cyberSW==1),]
  if(nrow(vv)>1) {
      up <- updf(site=qv$Site,cer.type=qv$Type,ct=qv$Count,start=qv$Begin,
                 end=qv$End,chron=qv$cyberSW,interval=25,
                 cutoff=0.25,min.period=10)
  }
  per.tab <- as.matrix(up$per.occ)
  if (length(per.tab)==2) {per.tab <- t(per.tab)}
  int.brk <- (per.tab[,2]-per.tab[,1])/interval
  tmp <- as.matrix(up$cer.occ[,7:ncol(up$cer.occ)])
  app <- matrix(0,nrow(tmp),sum(int.brk))
  app.per <- seq(min(up$per.occ),max(up$per.occ),by=interval)

  j <- 0
  for (m in 1:length(int.brk)) {
    j <- int.brk[m]+j
    app[,(j-int.brk[m]+1:int.brk[m])] <- tmp[,m]/int.brk[m]}
  full.app <- matrix(0,nrow(app),length(full.per)-1)
  colnames(full.app) <- paste('AD',full.lab[,1],sep='')

  if (min(up$per.occ)<beg.per) {
  app <-
    app[,min(which(app.per %in% seq(beg.per,end.per,by=interval))):ncol(app)]
  app.per <-
    app.per[-(1:min(which(app.per %in% seq(beg.per,end.per,by=interval)))-1)]}

  if (max(up$per.occ)>end.per) {
  app <- app[,1:(max(which(app.per %in% seq(beg.per,end.per,by=interval)))-1)]
  app.per <-
    app.per[-((max(which(app.per %in% seq(beg.per,end.per,by=interval)))+1):
                length(app.per))]}

  col.start <- which(full.lab[,1]==min(app.per))
```

```
  full.app[,col.start:(col.start+dim(app)[2]-1)] <- app
  out.cer[[i]] <- cbind(up$cer.occ[,1:6],full.app)

  out.cer
}

final <- do.call(rbind,out.cer)
```

Convert ceramic type data to wares using SQL expression

This function looks up ceramic information from cyberSW.org using a separate "Ceramic_type_master.csv" file.

```
Ceramic_type_master <-
  read.table(file = 'Ceramic_type_master.csv', sep = ',', header = T)
a.lab <- paste('AD', full.lab[, 1], sep = '')

sqltext <- 'SELECT final.Site, Ceramic_type_master.SWSN_Ware, '
for (i in 1:length(a.lab)) {
  sqltext <-
    paste(sqltext, "sum(final.", a.lab[i], ")*1 AS ", a.lab[i], ", ", sep =
            "")
}
sqltext <- substr(sqltext, 1, nchar(sqltext) - 2)
sqltext <-
  paste(
    sqltext,
    "FROM Ceramic_type_master INNER JOIN final ON
    Ceramic_type_master.SWSN_Type = final.Type
                WHERE (Ceramic_type_master.Decoration='bichrome' Or
    Ceramic_type_master.Decoration='polychrome' Or
    Ceramic_type_master.Decoration='undifferentiated dec') AND
    (Ceramic_type_master.SWSN_Ware Not Like 'Undiff%')
                GROUP BY final.Site, Ceramic_type_master.SWSN_Ware"
  )

wareapp <- sqldf(sqltext)
```

Save

Saves entire environment to RData file with the apportioned data so that this can be quickly retrieved without rerunning the UPDA function.

```
save.image('apportioned.RData')
```

Create ceramic networks

Load required libraries and saved environment (if necessary).

```
library(maptools)
library(sp)
require(sqldf)
require(compiler)
require(reshape)
require(msm)
require(snowfall)
require(parallel)
require(statnet)
library(igraph)
library(vegan)

load('apportioned.RData')
```

Setup intervals by combining consecutive 25 year apportioned intervals into 50-year intervals

```
wareapp2 <- wareapp

per.list2 <- c('AD1000_1050','AD1050_1100','AD1100_1150','AD1150_1200',
               'AD1200_1250','AD1250_1300','AD1300_1350','AD1350_1400',
               'AD1400_1450')
lookup.list <- list(c(31:32),c(33:34),c(35:36),
                    c(37:38),c(39:40),c(41:42),
                    c(43:44),c(45:46),c(47:48))

for (i in 1:length(per.list2)) {
  warevar <- wareapp2[,1:2]
  cer <- wareapp2[,lookup.list[[i]]]
  if (!is.null(ncol(cer))) {cer <- rowSums(cer)}
  cer <- cbind(warevar,cer)
  out <- cast(cer,Site~SWSN_Ware,fun.aggregate = sum)
  out[is.na(out)] <- 0
  rownames(out) <- out[,1]

  assign(paste(per.list2[i]),out)}
```

Trim and rename ceramic and attribute data frames by period

Note that due to site protection concerns, the locations of settlements used here have been jittered and differ from those used to generate the results presented in the article. Derived distance objects based on the real locations are used later in this script to ensure replication of the original results.

```
cer.output <- list(AD1000_1050,AD1050_1100,AD1100_1150,AD1150_1200,AD1200_1250,
                   AD1250_1300,AD1300_1350,AD1350_1400,AD1400_1450)

attr.comb <- read.table('attr_all.csv',header=T,sep=',')

for (i in 1:length(per.list2)) {
  cer <- (cer.output[[i]])
```

```
  cer <- cer[,-1]
  trim <- which(rowSums(as.matrix(cer))>=10)
  cer <- cer[trim,]
  cer2 <- as.matrix(cer[,which(colSums(cer)>0)])
  colnames(cer2) <- colnames(cer)[which(colSums(cer)>0)]
  row.names(cer2) <- row.names(cer)
    attr.temp <-
      attr.comb[which(as.vector(attr.comb[,2]) %in% row.names(cer2)),]
    cer2 <- cer2[which(row.names(cer2) %in% attr.temp[,2]),]
    attr.temp2 <- as.data.frame(matrix(NA,nrow(cer2),4))
    for (j in 1:nrow(attr.temp2)) {
      attr.temp2[j,] <-
        attr.temp[which(row.names(cer2)[j]==attr.temp[,2])[1],]
    }
    row.names(attr.temp2) <- row.names(cer2)
    attr.temp2[,1] <- row.names(attr.temp2)
  assign(paste((per.list2[i]),"cer",sep=""),cer2)
  assign(paste((per.list2[i]),"attr",sep=""),attr.temp2)}


coord.list <- list(AD1000_1050attr,AD1050_1100attr,AD1100_1150attr,
                   AD1150_1200attr,AD1200_1250attr,AD1250_1300attr,
                   AD1300_1350attr,AD1350_1400attr,AD1400_1450attr)

cer.list <- list(AD1000_1050cer,AD1050_1100cer,AD1100_1150cer,
                 AD1150_1200cer,AD1200_1250cer,AD1250_1300cer,
                 AD1300_1350cer,AD1350_1400cer,AD1400_1450cer)
```

Create similarity matrices and networks by period

```
sim.mat <- function(x) {
  names <- row.names(x)
  x <- na.omit(x)
  x <- prop.table(as.matrix(x),1)*100
  rd <- dim(x)[1]
  results <- matrix(0,rd,rd)
  for (s1 in 1:rd) {
    for (s2 in 1:rd) {
      x1Temp <- as.numeric(x[s1, ])
      x2Temp <- as.numeric(x[s2, ])
      results[s1,s2] <- 200 - (sum(abs(x1Temp - x2Temp)))}}
  row.names(results) <- names
  colnames(results) <- names
  results <- results/200
  results <- round(results,3)
  return(results)}


AD1000sim <- sim.mat(AD1000_1050cer)
AD1050sim <- sim.mat(AD1050_1100cer)
AD1100sim <- sim.mat(AD1100_1150cer)
```

```
AD1150sim <- sim.mat(AD1150_1200cer)
AD1200sim <- sim.mat(AD1200_1250cer)
AD1250sim <- sim.mat(AD1250_1300cer)
AD1300sim <- sim.mat(AD1300_1350cer)
AD1350sim <- sim.mat(AD1350_1400cer)
AD1400sim <- sim.mat(AD1400_1450cer)
```

Save

Saves entire environment to RData file

```
save.image('apportioned.RData')
```

Network breaks

Load required libraries and saved environment (if necessary). Set up for parallel processing.

```
library(statnet)
library(igraph)
library(deldir)
library(ggplot2)
library(FreeSortR)
library(parallel)
library(doParallel)
library(ecp)
library(reshape2)
library(ggpubr)
library(sf)
library(ggmap)

# set up cores for running in parallel
cl <- makeCluster(detectCores())
registerDoParallel(cl)
```

Calculate network breaks looping over each period

```
# assign attribute dataset and similarity matrix objects
name.list <- c('1000-1050 CE','1050-1100 CE','1100-1150 CE','1150-1200 CE',
               '1200-1250 CE','1250-1300 CE','1300-1350 CE','1350-1400 CE',
               '1400-1450 CE')


coord.list <- list(AD1000_1050attr,AD1050_1100attr,AD1100_1150attr,
                   AD1150_1200attr,
                   AD1200_1250attr,AD1250_1300attr,AD1300_1350attr,
                   AD1350_1400attr,
                   AD1400_1450attr)
```

```
sim.list <- list(AD1000sim,AD1050sim,AD1100sim,AD1150sim,AD1200sim,
                 AD1250sim,AD1300sim,AD1350sim,AD1400sim)

# create empty lists for results
proto.list <- list()
breaks.list <- list()
fig.list <- list()
res.list <- list()


# Loop over each of 9 intervals
for (m in 1:9) {
  # create temporary object names for locations and sim matrix
  coord1 <- coord.list[[m]][, 3:4]
  sim1 <- sim.list[[m]]

  # convert UTM locations into Euclidean distance matrix and reverse direction
  d <- as.matrix(dist(coord1))

  # Calculate weighted Louvain clustering for the selected similarity
  # network constrained across every percentile in the distance matrix
  res <- matrix(NA, nrow(sim1), 100)
  for (i in 1:100) {
    d.mat <- event2dichot(d, method = 'quantile', thresh = 1 - (i / 100))
    com.net <- d.mat * sim1
    G1 <-
      graph.adjacency(com.net,
                      mode = "undirected",
                      weighted = TRUE,
                      diag = TRUE)
    set.seed(63246)
    res[, i] <- cluster_louvain(G1, weights = E(G1)$weight)$membership
  }
  res.list[[m]] <- res

  # Calculate Adjusted Rand Index among every pair of percentile
  # constrained partition sets
  rand.I <- foreach(i = 1:100) %dopar% {
    res2 <- NULL
    for (j in 1:100) {
      res2[j] <- FreeSortR::RandIndex(res[, i], res[, j])$AdjustedRand
    }
    res2
  }
  rand.I <- matrix(unlist(rand.I), ncol = 100, byrow = TRUE)
  rand.I[is.na(rand.I)] <- 0 # assign any NA comparisions 0

  ## define breakpoints using aggolomerative algorithm in ecp package
  ## first across all distances
  z1 <-
    e.agglo(
      rand.I,
      alpha = 1,
```

```
    penalty = function(cp, Xts)0) # calculate across all distances
  z1.a <- z1$estimates - 1
  z2 <-
    e.agglo(
      rand.I[1:25, 1:25],
      alpha = 1.5,
      penalty = function(cp, Xts)0) # calculate within the lowest quartile
  z2.a <- z2$estimates - 1
  z2.a <- z2.a[-length(z2.a)]
  # combine and remove duplicates
  z.a <- sort(unique(c(z1.a, z2.a)))
  z.a[1] <- 1
  z.a <- unique(z.a)

  #calculate prototypical distances for each partition and create data frame
  proto.typical <- NULL
  for (i in 1:(length(z.a) - 1)) {
    lv <-  z.a[i]:z.a[i + 1]
    proto.typical[i] <-  lv[which.max(rowSums(rand.I[lv, lv]))]
  }
  z.p <- as.data.frame(proto.typical)
  colnames(z.p) <- c('X')


  ## Plot Rand Index similarities with proto typical distances
  ## and partitions indicated
  fig.list[[m]] <- ggplot() +
    scale_fill_viridis_c() +
    geom_tile(data = melt(rand.I), aes(x = Var1, y = Var2, fill = value)) +
    geom_vline(xintercept = z.a + 0.5, col = "white") +
    geom_hline(yintercept = z.a + 0.5, col = "white") +
    geom_point(
      data = z.p,
      mapping = aes(x = X, y = X),
      size = 4,
      col = 'red'
    ) +
    ggtitle(label = name.list[[m]]) +
    xlab("") +
    ylab("")


  ## Remove boundary for lower quartile
  z.a2 <- z.a[-1]
  z.a2 <- z.a2[-length(z.a2)]

  ## Record breakpoints and prototypical distances in kilometers
  proto.list[[m]] <- quantile(d / 1000, z.p / 100)
  breaks.list[[m]] <- quantile(d / 1000, z.a2 / 100)
}
```

Create data frame containing prototypical distances, period, and break number

```
time.list <- c("1000", "1050", "1100", "1150", "1200", "1250",
               "1300", "1350", "1400")
vv <- as.data.frame(unlist(proto.list))
time.per <- NULL
seq.per <- NULL
for (m in 1:9) {
  temp1 <- rep(time.list[m], length(proto.list[[m]]))
  temp2 <- seq(1:length(proto.list[[m]]))
  time.per <- c(time.per, temp1)
  seq.per <- c(seq.per, temp2)
}
vv <- cbind(vv, time.per, seq.per)
colnames(vv) <- c("prototypical", "Period", "Num")
```

Figure 2

```
base = get_map(location=c(-116,28.5,-101.5,39), zoom=6,
               maptype="terrain-background", color="bw")

group <- 6 # Set which period group you want to plot

d1 <- as.matrix(dist(coord.list[[m]][,3:4]))

d1 <- d1/1000

df <- as.data.frame(as.vector(d1))

dist.group <- c(1,0.9,0.5,0.1)
dist.thresh <- as.vector(quantile(d1,dist.group))
dist.title <- c("100%","90%","50%","10%")
dist.group2 <- c(100,90,50,10)

hist.list <- list()

for (m in 1:4) {
hist.list[[m]] <- ggplot(df, aes(x=as.vector(d1))) +
  geom_histogram(colour="black", fill="lightgray") +
  annotate("rect", xmin=-Inf, xmax=dist.thresh[m],
           ymin=0, ymax=Inf, alpha=0.25, fill="blue") +
  theme(plot.title = element_text(size=14, face="bold.italic"),
        panel.grid.major.y = element_line('darkgray', size=0.25),
        panel.grid.minor.y=element_blank(),
        panel.grid.major.x=element_blank(),
        panel.grid.minor.x=element_blank(),
        panel.background=element_blank(),
        axis.line = element_line(color='black', size = 1, linetype = "solid"),
        text=element_text(size=14)) +
  ggtitle(dist.title[m]) +
  scale_x_continuous(name="Kilometers")
```

```
}


hist.map.list <- list()

for (k in 1:4) {

locations_sf <- st_as_sf(coord.list[[group]], coords = c("V3", "V4"), crs = 26912)
z <- st_transform(locations_sf,crs=4326)

coord1 <- do.call(rbind, st_geometry(z)) %>%
  tibble::as_tibble() %>% setNames(c("lon","lat"))

df <- as.data.frame(coord1)

hist.map.list[[k]] <- ggmap(base) +
  geom_point(data=df, aes(x=lon,y=lat),
             color=res.list[[group]][,dist.group2[k]], size=2) +
  scale_color_brewer(palette="Set2") +
  ggtitle(label=paste(name.list[group],", K = ",
                      length(table(res.list[[group]][,dist.group2[k]])),
                      sep="")) +
  theme_minimal() +
  xlab("") +
  ylab("") +
  theme(axis.text.x=element_blank(), #remove x axis labels
        axis.ticks.x=element_blank(), #remove x axis ticks
        axis.text.y=element_blank(),  #remove y axis labels
        axis.ticks.y=element_blank()  #remove y axis ticks
  )
}


ggarrange(hist.list[[1]],hist.list[[2]],hist.list[[3]],hist.list[[4]],
          hist.map.list[[1]],hist.map.list[[2]],hist.map.list[[3]],
          hist.map.list[[4]],
          nrow=2,ncol=4)
```

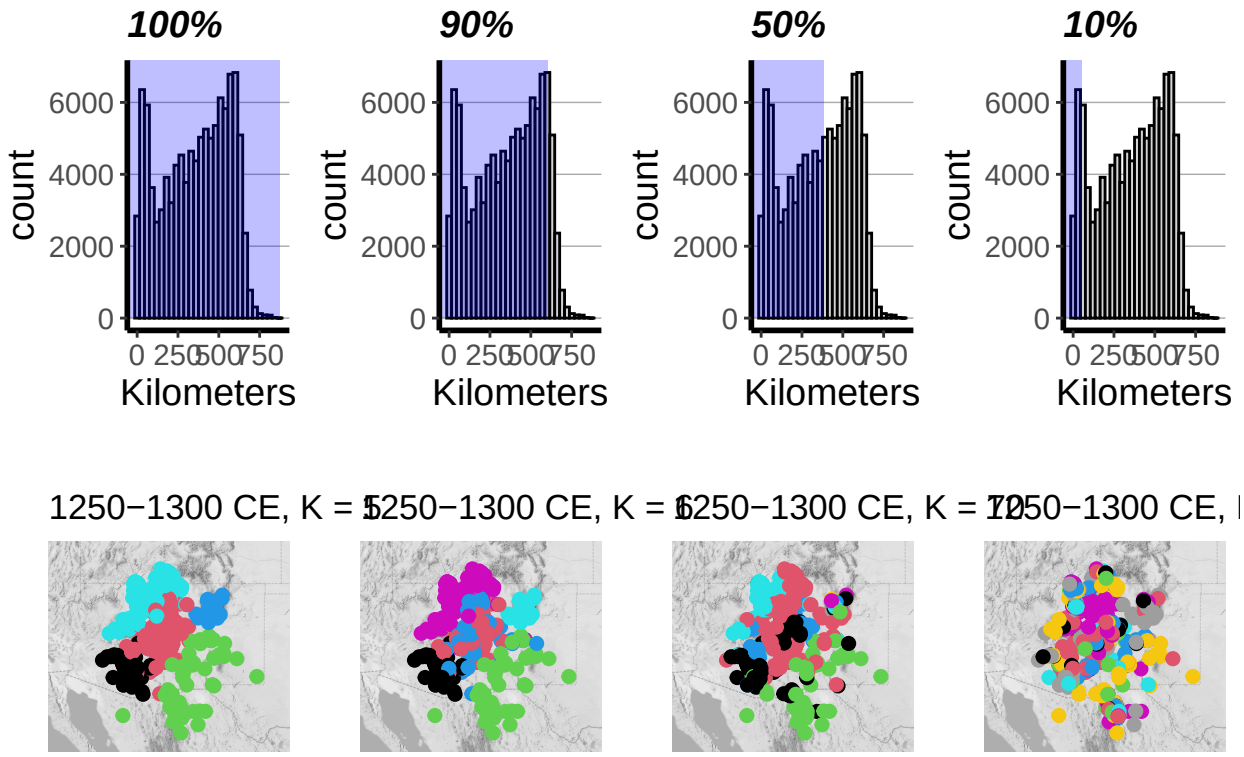


Figure 3

```
fig.list[[9]] # defined within the Network Breaks section
```

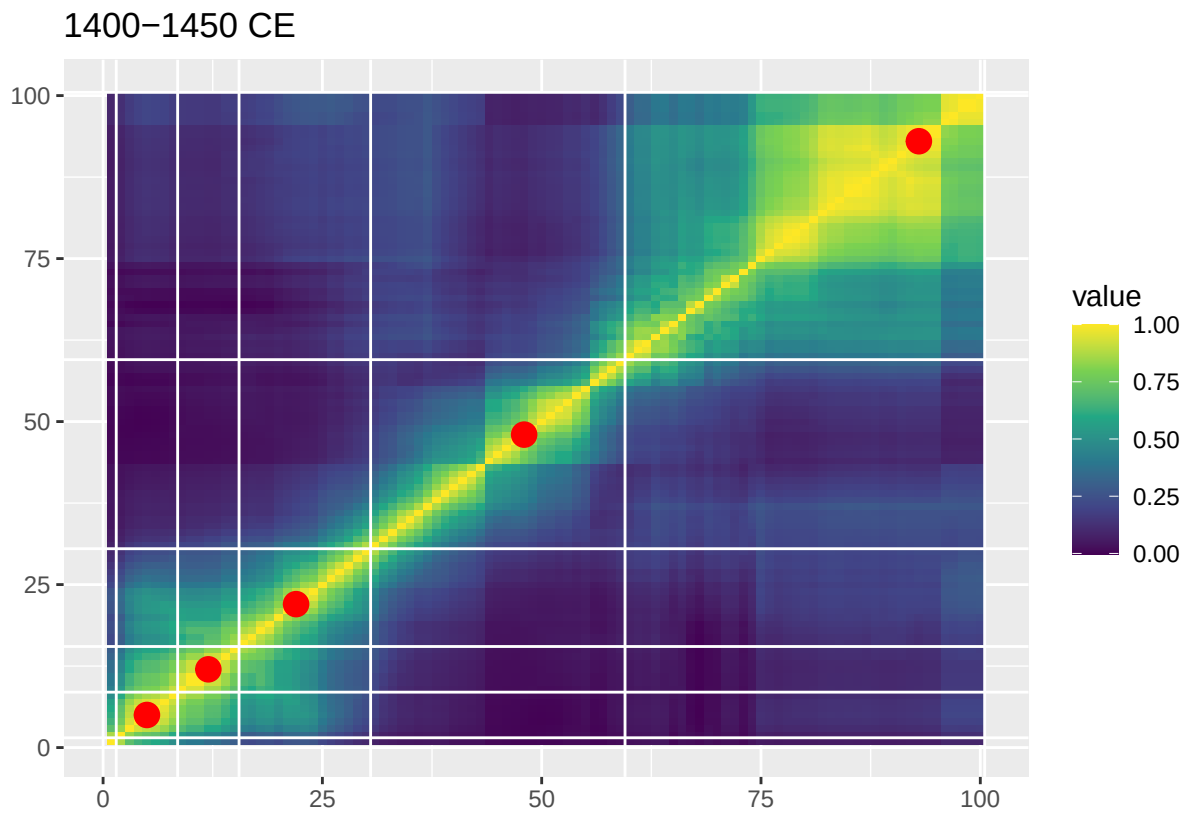


Figure 4

```
vv$Num = factor(vv$Num)
ggplot(data=vv, aes(x = Num, y=prototypical, group=Num, fill=as.factor(Num))) +
  geom_boxplot() +
  ylab("Distance") +
  xlab("Partition Group") +
  guides(fill = "none") +
  theme(text=element_text(size=20), axis.title.x=element_blank()) +
  theme_bw()
```

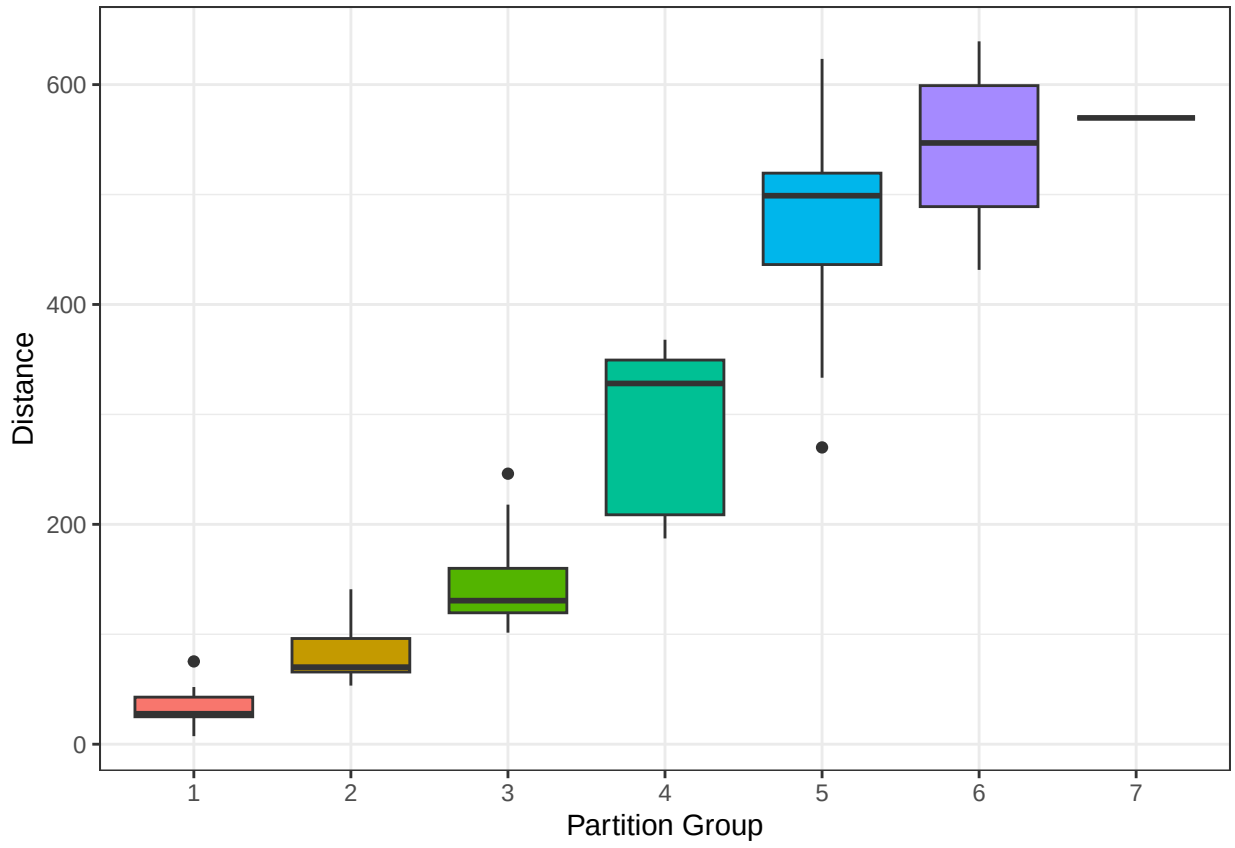


Figure 5

```
ggplot(data=vv, aes(x=Period,y=prototypical, group=Num, color=as.factor(Num))) +
  geom_line(lwd = 1.15) +
  geom_point(size = 2.5) +
  ylab("Distance") +
  labs(color = "Partition Group") +
  theme_bw()
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
```

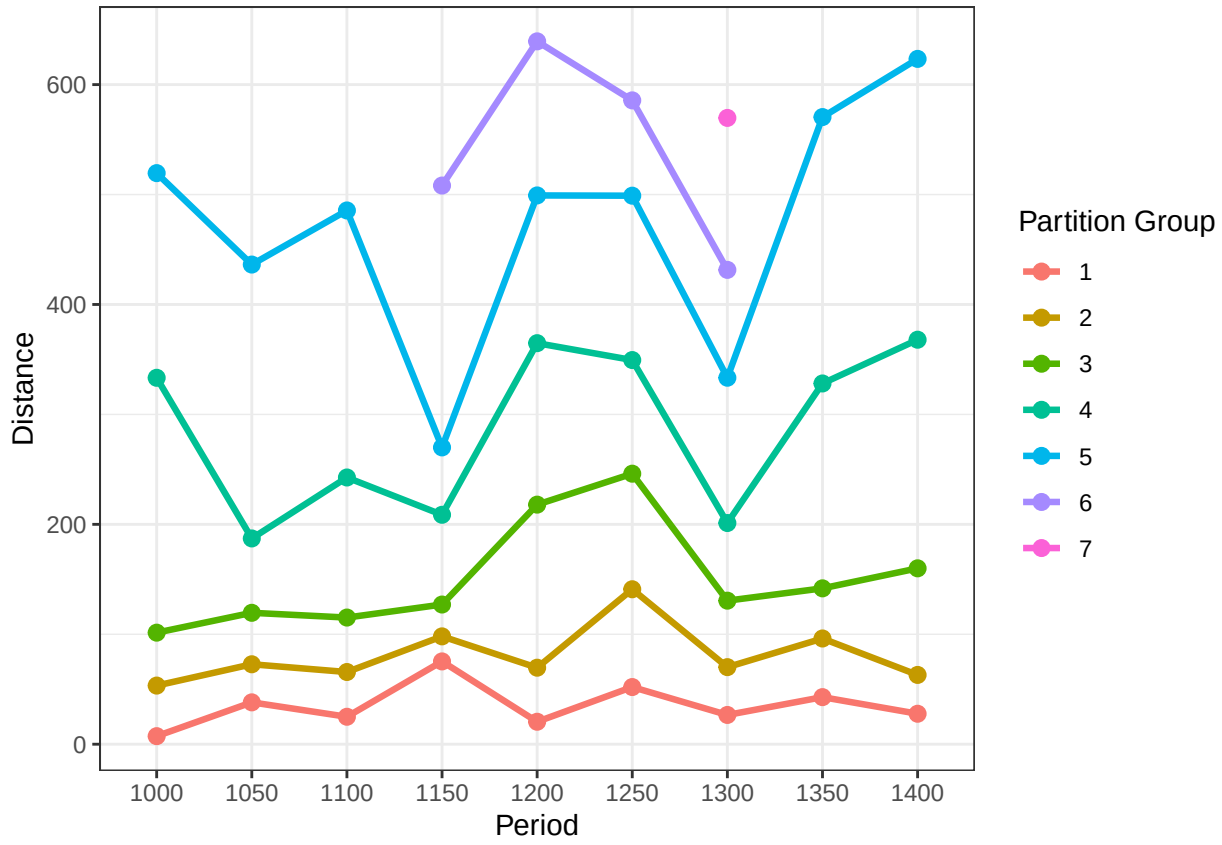
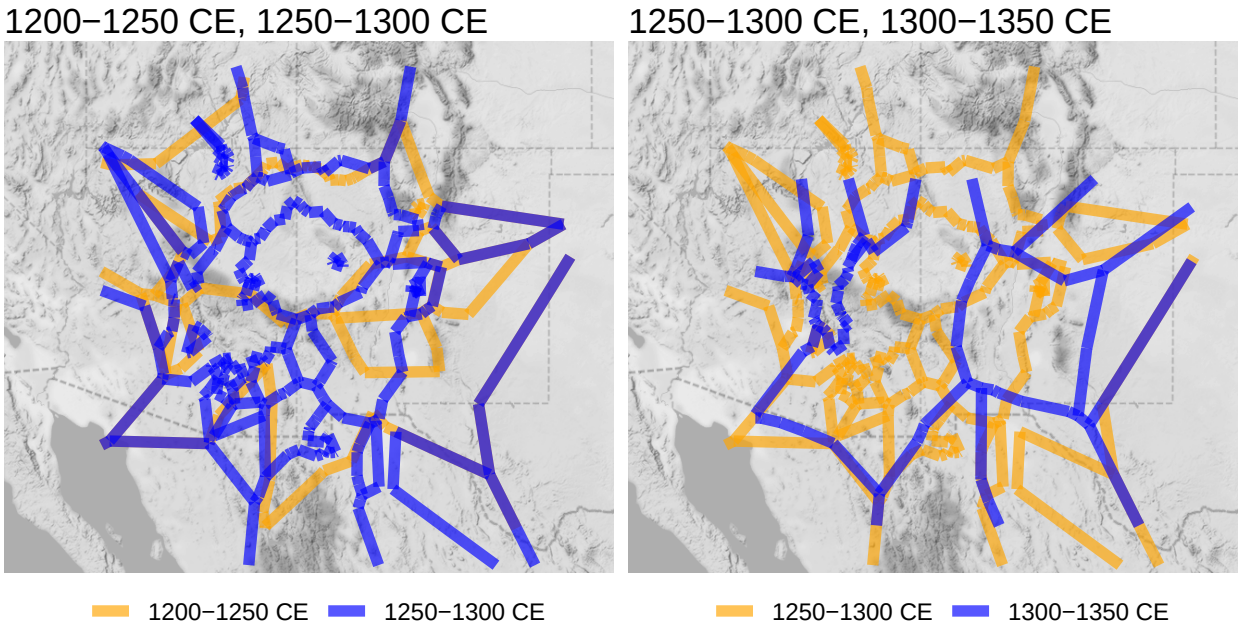


Figure 6

```
library(ggmap)
library(sf)
library(tibble)
library(deldir)

# Create output object and download basic background map from ggmap
map.list <- list()
dd2.list <- list()
base = get_map(location=c(-116,28.5,-101.5,39), zoom=6,
               maptype="terrain-background", color="bw")


# Loop over each period
for (m in 1:length(name.list)) {

  ## Create and transform sf coordinates for locations
  locations_sf <- st_as_sf(coord.list[[m]], coords = c("V3", "V4"), crs = 26912)
  z <- st_transform(locations_sf,crs=4326)
  coord1 <- do.call(rbind, st_geometry(z)) %>%
    tibble::as_tibble() %>% setNames(c("lon","lat"))

  ## Define the column number for the appropriate prototypical distance
```

```
  grp <- as.numeric(gsub("%", "", names(proto.list[[m]][2])))

  # Remove groups containing only a single node and create new data frame
  df <- as.data.frame(coord1)
  res2 <- res.list[[m]][,grp]
  res.lk <- which(table(res2)>1)
  res2a <- res2[which(res2 %in% res.lk)]
  df2 <- df[which(res2 %in% res.lk),]
  df2 <- na.omit(df2)

  # use Delaunay Triangulation to create Voronoi Tessalation around nodes
  dd <- deldir(df2$lon,df2$lat,z=as.factor(res2a))
  # Create dividing chain around sites for junctures and assign to list object
  dd2 <- divchain(dd)
  dd2.list[[m]] <- dd2

  # Plot map with Voronoi boundaries
  map.list[[m]] <- ggmap(base) +
    geom_point(data=df, aes(x=lon,y=lat), color=res2, size=2) +
    geom_segment(aes(x = x0, y = y0, xend = x1, yend = y1),
                 size = 1.5, data = dd2.list[[m]],
                 linetype = 1, color= "black") +
    ggtitle(label=name.list[m])
}

# Create map object showing consecutive pairs of boundaries
map.group <- list()
for (i in 1:8) {
  ddlines = rbind(dd2.list[[i]] %>% mutate(period = name.list[i]),
                  dd2.list[[i+1]] %>% mutate(period = name.list[i + 1]))
map.group[[i]] <- ggmap(base) +
  geom_segment(aes(x = x0, y = y0, xend = x1, yend = y1, color = period),
               size = 2, data = ddlines, linetype = 1, alpha = .66) +
  scale_color_manual(values = c("orange","blue")) +
  ggtitle(label=paste(name.list[i], ", ", name.list[i+1], sep="")) +
  xlab("") +
  ylab("") +
  theme_void() +
  theme(legend.position = 'bottom', legend.title = element_blank(),
        plot.margin = unit(c(1,1,1,1), 'mm'))
}

library(ggpubr)

ggarrange(map.group[[5]],map.group[[6]],nrow=1)
```



Examine overlap between boundaries for each period

```
library(raster)
library(dplyr)
library(reshape2)

# Set up lists for output
z.list <- list()
v.list <- list()

# Loop across all intervals
for (m in 1:length(name.list)) {
  # defome data frame from Voronoi boundary object
  df <-  as_tibble(dd2.list[[m]])

  # convert object into st and create a buffer of 18000 meters
  rows <- split(df, seq(nrow(df)))
  lines <- lapply(rows, function(row) {
    lmat <- matrix(unlist(row[1:4]), ncol = 2, byrow = TRUE)
    st_linestring(lmat)
  })
  lines <- st_sfc(lines)
  lines_sf <- st_sf('geometry' = lines,crs=4326)
  z.list[[m]] <- lines_sf %>% st_transform(crs=26912)
```

```
  v.list[[m]] <- st_buffer(lines_sf,18000) %>%
    st_transform(crs=26912)
}
names(z.list) = name.list
names(v.list) = name.list


# generate raster of common boundaries and project
boundaries = do.call(rbind,z.list)
boundaries = st_transform(boundaries,4326)
boundary = pgirmess::bbox2sf(bbox=st_bbox(boundaries))
r = raster(ncol = 500, nrow = 500)
bbox = st_bbox(boundary) %>% as.vector
bbox = bbox[c(1,3,2,4)]
r = setExtent(r, bbox)
crs(r) = "+proj=longlat +datum=WGS84 +no_defs +type=crs"
r[] = 1


# apply buffers to raster using mask function
res.mat <- matrix(NA,9,9)
for(i in 1:length(v.list)){
  for(j in 1:length(v.list)) {
    rmask = list()
    rmask[[1]] = raster::mask(r,v.list[[i]] %>% st_transform(4326))
    rmask[[2]] = raster::mask(r,v.list[[j]] %>% st_transform(4326))

    rbrick = stack(rmask)

    rCombind = calc(rbrick,fun = sum, na.rm = T)

    # look only at common boundaries
    rCommon = rCombind

    # Create dataframe of boundary overlap
    overlapDF = raster::as.data.frame(rCommon, na.rm = T) %>%
      count(layer)
    overlapDF <- overlapDF[-1,]
    # Calculate degree of overlap
    res.mat[i,j] = overlapDF[which(overlapDF$layer==2),2] / sum(overlapDF$n)
  }
}

diag(res.mat) <- NA

temp <- NULL
for (i in 1:8) {
  temp[i] <- res.mat[i,i+1]
}

res.mat2 <- res.mat
res.mat2[which(res.mat %in% temp)] <- NA
temp2 <- res.mat2[upper.tri(res.mat2)]
```

```
temp2 <- temp2[!is.na(temp2)]


res.bp <- as.data.frame(rbind(cbind(temp,rep("T+1",length(temp))),
                              cbind(temp2,rep("T+2..K",length(temp2)))))
res.bp$temp <- as.numeric(res.bp$temp)
res.bp$V2 <- as.factor(res.bp$V2)
```

Figure 7

```
ggplot(data=res.bp, aes(y=temp, x=V2, fill=V2), color=as.factor(V2)) +
  geom_boxplot() +
  ylab("Proportion Overlap") +
  xlab("Temporal Group")+
  labs(fill = "Temporal Group") +
  theme_bw(base_size=20)
```

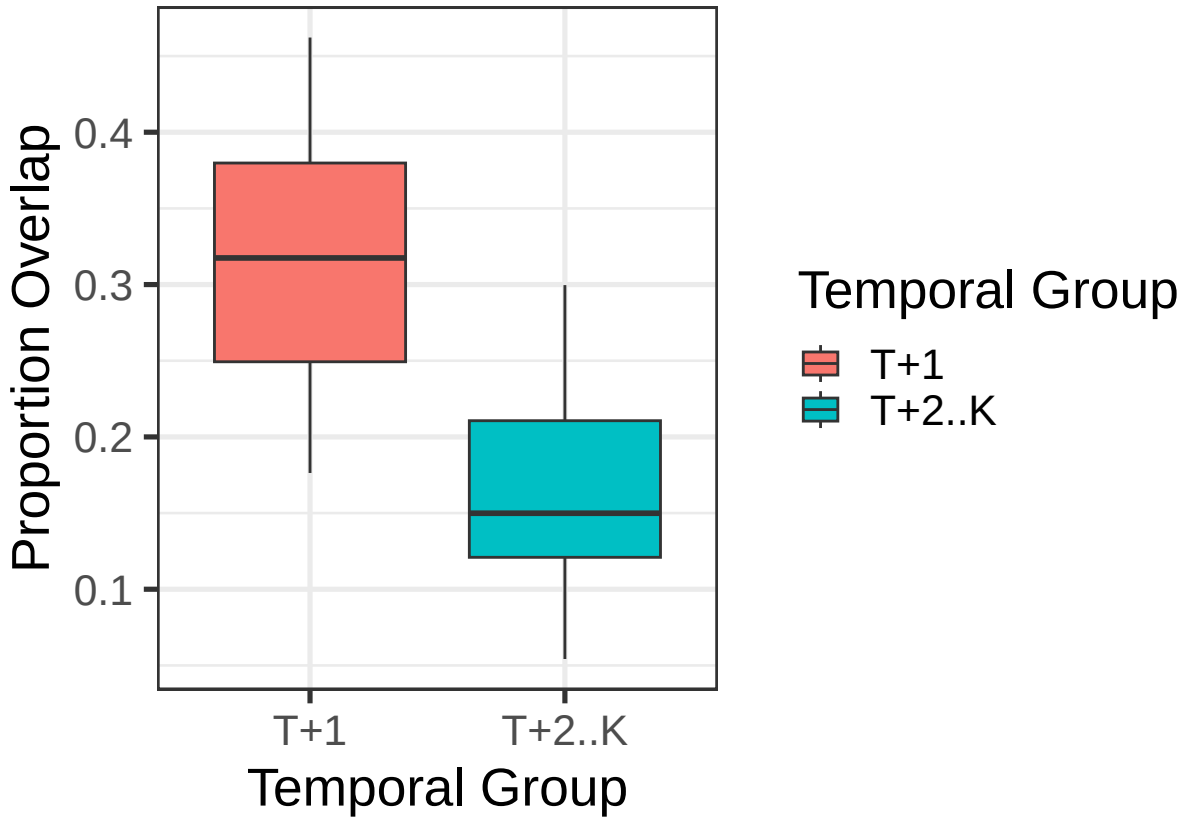


Figure 9

Note this function takes a long time and is not run in this knitted document.

```
library(tidyverse)
library(sf)
library(raster)
library(ggfx)
library(FedData)
library(ggnewscale)
library(ggsn)
library(colorspace)

rmask2 = lapply(v.list, function(l)
  raster::mask(r, l %>% st_transform(4326)))
rbrick = stack(rmask2)
rCom = calc(rbrick, fun = sum, na.rm = T)

rC1 = rC2 = rCom
rC1[rC1 < 3] <- NA
rC2[rC2 < 6] <- NA

rC1df = rC1 %>% as.data.frame(xy = T) %>%
  filter(!is.na(layer)) %>%
  mutate(layer = factor(layer, levels = 3:9))
rC2df = rC2 %>% as.data.frame(xy = T) %>%
  filter(!is.na(layer)) %>%
  mutate(layer = factor(layer, levels = 3:9))

locations_sf <-
  st_as_sf(attr.comb2,
           coords = c("EASTING", "NORTHING"),
           crs = 26912)

buffer = locations_sf %>% st_buffer(100000)
ned = get_ned(buffer, "ned")
ned = aggregate(ned, 12, max)
sl <- terrain(ned, "slope", unit = "radians")
asp <- terrain(ned, "aspect", unit = "radians")
hs = hillShade(sl, asp, angle = 40, direction = 270)
hsdf = hs %>%
  raster::as.data.frame(xy = T) %>%
  rename(v = layer) %>%
  mutate(v = cut(v, 21))
neddf = ned %>%
  raster::as.data.frame(xy = T) %>%
  rename(v = layer) %>%
  mutate(v = cut(v, 11))
col1 = divergingx_hcl(
  n = 5,
  h1 = 80.76,
  h2 = 80,
  h3 = 60,
  c1 = 19.36,
  cmax1 = 50,
  c2 = 35,
  c3 = 55,
```

```
  l1 = 96.52,
  l2 = 84,
  l3 = 77,
  p1 = 1.1,
  p3 = 1.0
)
col2 = divergingx_hcl(
  n = 6,
  h1 = 64,
  h2 = 148,
  h3 = 80,
  c1 = 60,
  cmax1 = 60,
  c2 = 21,
  c3 = 17,
  l1 = 85,
  l2 = 31.5,
  l3 = 79,
  p1 = 1.1,
  p3 = 1.0
)
cols = c(col1, col2) %>% desaturate(amount = .4)
rivers = readRDS("GIS/rivers.Rds")
lakes = st_read("GIS/lakes.geojson")
oceans = st_read("GIS/oceans.geojson")
political = st_read("GIS/political.geojson")
base = ggplot() +
  with_blur(
  geom_raster(
    data = neddf,
    aes(x, y, fill = v),
    interpolate = T,
    na.rm = T
  ),sigma = unit(.2, 'mm')
  ) +
  scale_fill_manual(name = 'v',
                    values = cols,
                    guide = 'none') +
  new_scale('fill') +
  geom_raster(
    data = hsdf,
    aes(x, y, fill = v),
    alpha = .5,
    interpolate = T
  ) +
  scale_fill_grey(guide = 'none', na.value = 'white') +
  geom_sf(data = oceans, fill = "lightblue") +
  with_blur(
  geom_sf(data = rivers,
          color = "lightblue",
          alpha = .5),sigma = unit(.2, 'mm')
  ) +
  with_blur(
```

```
  geom_sf(
    data = lakes,
    color = "transparent",
    fill = "lightblue",
    alpha = .5
  ),sigma = unit(.2, 'mm')
  )+
  geom_sf(data = political,
          color = "#404040",
          fill = "transparent") +
  geom_sf(
    data = locations_sf,
    color = '#404040',
    size = 1,
    alpha = .5) +
  north(
    x.min = -113.65,
    x.max = -100.65,
    y.min = 29,
    y.max = 40.44,
    location = "bottomleft",
    symbol = 14
  ) +
  scalebar(
    x.min = -114.15,
    x.max = -102,
    y.min = 29.4,
    y.max = 40.44,
    location = "bottomright",
    dist = 100,
    dist_unit = "km",
    transform = T,
    model = "WGS84",
    st.size = 2,
    border.size = .5
  ) +
  theme_void() +
  coord_sf(
    xlim = c(-114.15, -100.65),
    ylim = c(28.9, 40.44),
    crs = 4326,expand = F,clip = 'on'
  ) +
  theme(legend.position = 'bottom',plot.margin = unit(c(1,1,1,1),"mm"))

plasma = c(
  "#DAFF47FF",
              "#ECC000FF",
              "#E8853AFF",
              "#D24E71FF",
              "#AB1488FF",
              "#72008DFF",
              "#001889FF"
)
```

```
g1 = base +
  new_scale('fill') +
  with_blur(geom_raster(data = rC1df, aes(
    x = x, y = y, fill = layer
  ), alpha = .9),
  sigma = unit(.4, 'mm')) +
  scale_fill_manual(values = plasma) +
  guides(fill = guide_legend(title = "n of periods")) +
  ggtitle("")
g2 =  base +
  new_scale('fill') +
  with_blur(geom_raster(data = rC2df, aes(
    x = x, y = y, fill = layer
  ), alpha = .9),
  sigma = unit(.4, 'mm')) +
  scale_fill_manual(values = plasma[4:7]) +
  guides(fill = guide_legend(title = "n of periods"))
```