Creating Migratory Networks in R

Matt DeSaix

2022-03-08

Contents

Chapter 1

Introduction

This text outlines how to create migratory networks using the R package `Mignette` (**Mig**ratory **net**work **t**ools **e**nsemble). The creation of a migratory network is broken into three main parts:

1. Delineating breeding nodes
2. Delineating wintering nodes
3. Create migratory network

1.1 Installation

You can install the development version of Mignette from GitHub with:

```
# install.packages("remotes")
remotes::install_github("mgdesaix/Mignette", auth_token = "xxx")
```

Note: The `auth_token =` specifies my personal access token to allow the download because it is currently a private repository. Replace "xxx" with the token until we make the repository public.

Chapter 2

Breeding nodes

Breeding nodes are delineated by the genetically distinct populations on the breeding grounds. In this example, we'll show how to use eBird Status and Trends data to specify the breeding range and then use genetic data from admixture analyses to specify the spatial extent of the breeding nodes.

2.1 ebirdst

In the migratory network analyses, the `ebirdst` abundance data is used to delineate the different stages of the annual cycle. The `Mignette` package provides code for these first sections that function as a wrapper for the `ebirdst` package to streamline the process. If you would like to know more about the underlying `ebirdst` code and analyses, check out the excellent tutorial by Strimas-Mackey, Auer, and Fink.

Prior to doing anything with eBird Status and Trends data, you will need to download the `ebirdst` package, and then get access to the data. To download the package:

```
# install.packages("remotes")
remotes::install_github("CornellLabofOrnithology/ebirdst")
```

Then, get access to `ebirdst` data at https://ebird.org/st/request. You will receive a key to download `ebirdst` data and you can enter that key in R:

```
ebirdst::set_ebirdst_access_key("XXXXX")
```

where `"XXXXX"` is the key.

The primary packages for this vignette are:

```
library(Mignette)
library(sf)
library(terra)
library(tidyverse)
library(ebirdst)
```

2.2 Seasonal abundance

2.2.1 WARNING: This section is currently deprecated. Please use the ebirdst tutorial to generate seasonal abundance polygons and continue to section 2.4 below

The first function, `get_ebirdst_abd_season()`, downloads species data and creates a multi-layered raster of seasonal abundance data (nonbreeding, prebreeding migration, breeding, and postbreeding migration). The function currently takes two inputs, `species` and `path`. `ebirdst` data download is based on the six-letter species code, thus, we use the same naming system. You can find the specific codes with the get_species function from the `ebirdst` package.

Specify the species of interest for `get_ebirdst_abd_season()` with `species`.

Below is an example for downloading data for the Common Yellowthroat and creating the seasonal abundance raster stack.

```
# This can take a while depending on the species (5-10 min.)
abd_season <- get_ebirdst_abd_season(species = "comyel")
```

2.3 Generating seasonal polygons

2.3.1 WARNING: This section is currently deprecated. Please use the ebirdst tutorial to generate seasonal abundance polygons and continue to section 2.4 below

Using the previously created seasonal abundance rasters, we will convert them to polygons of the range. The details can be found in the `ebirdst` tutorial, but the gist of it is that we'll distinguish non-zero abundance from non-predicted areas, and delineate nice *smooth* ranges for the different stages.

We also need land extent data. We will get land data from the `rnaturalearth` package using the following code. Depending on the organism's range, you will

need to filter continent to the appropriate region. Here, we want both North and South America.

```
ne_scale <- 50
# land polygon
ne_land <- rnaturalearth::ne_countries(scale = ne_scale, returnclass = "sf") %>%
  dplyr::filter(continent %in% c("North America", "South America")) %>%
  sf::st_set_precision(1e6) %>%
  sf::st_union() %>%
  sf::st_geometry()
ne_land_proj <- sf::st_transform(ne_land, crs = sf::st_crs(abd_season))
```

Now we have all the input data we need to get the polygons of the range. The function `range_smooth()` in the `Mignette` package takes care of this. In the process of smoothing the polygon, small regions are dropped and holes in the polygon filled in based on the size (km^2) specified with the `smooth_area` parameter. Below it is set at 1000 km^2 (e.g. a 31.6 km * 31.6 km square), but you may want to increase or decrease depending on the organism.

```
# this fuction can take a while (~10 min.)
comyel_range_smooth <- range_smooth(abd_season = abd_season,
                                    ne_land = ne_land_proj,
                                    smooth_area = 1000,
                                    split_migration = FALSE,
                                    show_yearround = FALSE)
```

Extracting a single polygon of a portion of the range is simple and quick! Here's an example of getting the *breeding* season range from the `range_smooth()` output.

```
comyel_breed_smooth <- dplyr::filter(comyel_range_smooth,
                                     season == "breeding") %>%
                       sf::st_transform(crs = 4326)

sf::st_write(comyel_breed_smooth, dsn = './comyel/shapefiles',
          layer = "comyel_breed_smooth",
          driver = "ESRI Shapefile")
```

2.4 Creating the genoscape

This is modified from Eric Anderson's Github project that uses a matrix of individual Q-values to create a rasters of genetically distinct clusters - the **genoscape**. If you want to learn the ins and outs of making a *beautiful*

genoscape map, check out Eric's awesome tutorial. We will use the breeding polygon created in the previous step to specify the breeding range for the genoscape. The input data we need are:

- Individual Q-value matrix
- Lat/lon matrix of individual
- Breeding range polygon

The `comyel_breeding_data` data set provides admixture results (Q-values) of five genotype clusters for Common Yellowthroat (cite a coye paper) and metadata for the sampled individuals.

```
Q_matrix <- Mignette::comyel_breeding_data %>%
  dplyr::select(CA, Midwest, NewEngland, West, Southwest) %>%
  as.matrix()
long_lat_matrix <- Mignette::comyel_breeding_data %>%
  dplyr::select(Long, Lat) %>%
  as.matrix()
cluster_colors <-  c(
  CA = "#CC0000",
  Midwest = "#3399FF",
  NewEngland = "#9933CC",
  West = "#009933",
  Southwest = "#FF6600")
```

We will use a modified version of the `tess3r` package to create the genoscape rasters.

```
# remotes::install_github("eriqande/TESS3_encho_sen")
genoscape_brick <- tess3r::tess3Q_map_rasters(
  x = Q_matrix,
  coord = long_lat_matrix,
  map.polygon = breed_smooth,
  window = terra::ext(breed_smooth)[1:4],
  resolution = c(300,300), # if you want more cells in your raster, set higher
  col.palette = tess3r::CreatePalette(cluster_colors, length(cluster_colors)),
  method = "map.max",
  interpol = tess3r::FieldsKrigModel(10),
  main = "Ancestry coefficients",
  xlab = "Longitude",
  ylab = "Latitude",
  cex = .4
)
names(genoscape_brick) <- colnames(Q_matrix)
```

```
out.files <- paste0("./comyel/genoscape/comyel_genoscape_cluster_", names(genoscape_brick), ".tif
terra::writeRaster(terra::rast(genoscape_brick), filename = out.files)
```

2.5 Genoscape polygons

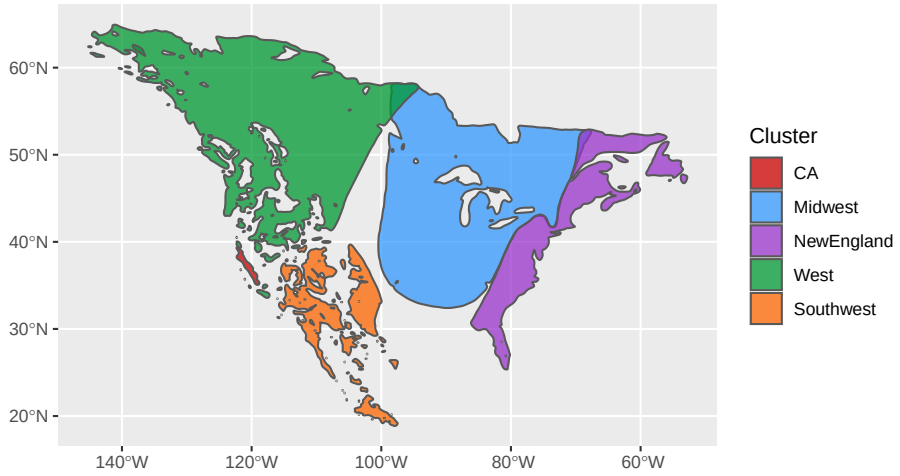
Using the genoscape rasters we will convert them to polygons, using the handy `scape_to_shape()` function. The `prob_threshold` parameter specifies the value to determine if a raster cell is included in the polygon for that genoscape. This value should be customized for different species to check for overlap of genoscape polygons, which is not desirable. Setting too high of a threshold will create very small breeding nodes, while too low of a threshold will result in large, overlapping breeding nodes.

```
genoscape_files <- list.files("./comyel/genoscape",
                                    full.names = T,
                                    pattern = "*.tif")

genoscape_raster_stack <- terra::rast(genoscape_files)
genoscape_polygon_sf <- scape_to_shape(x = genoscape_raster_stack, prob_threshold = 0.5)
```

Check out the polygons

```
ggplot() +
  geom_sf(data = genoscape_polygon_sf,alpha = 0.75, aes(fill = Cluster)) +
  scale_fill_manual(values = cluster_colors)
```



Check which polygons are overlapping. Each row of the output provides a pair of overlapping polygons (if there are any).

```
check_genoscape_overlap(genoscape_polygon_sf)
```

```
##      [,1]       [,2]
## [1,] "CA"       "West"
## [2,] "Midwest"  "NewEngland"
## [3,] "Midwest"  "West"
```

Now on to creating the wintering nodes

Chapter 3

Wintering nodes

For the migratory networks, we will use ecoregions to define the nonbreeding nodes. However, other nonbreeding nodes could be defined by the user instead. If you already have polygons defining your nonbreeding of nodes interest, then move along to …

3.1 Subsetting winter ecoregions

The ecoregion data is provided by [provide link]. The ecoregions are provided in this this package as `ecoregions_simple`. We will intersect the ecoregions with the wintering range of the Common Yellowthroat to identify all the ecoregions that overlap with the wintering range.

```
comyel_winter <- comyel_range_smooth %>%
  dplyr::filter(season == "nonbreeding") %>%
  sf::st_transform(crs = 4326) %>%
  sf::st_intersection(MuSpTest::ecoregions_simple) %>%
  dplyr::select(Region)
```

3.2 Snap points

Sometimes individuals are not quite within the wintering nodes. Here, we will make sure all sampled individuals get assigned to the nearest ecoregion.

```
winter.coords <- MuSpTest::comyel_wintering_data %>%
  st_as_sf(coords=c("Long","Lat")) %>%
  st_cast("MULTIPOINT") %>%
```

```
  st_set_crs(4326)

new.winter.coords <- snap_points(winter.coords, comyel_winter, 150000)
```

3.3 Finalize wintering nodes

Now that points have been *snapped* to the appropriate ecoregions, we can further subset all of the ecoregions in which we have actually sampled individuals from. If we haven't sampled individuals from a region, we can't use that region as a node in the migratory connectivity network!

```
winter_intersect <- st_intersects(comyel_winter, new.winter.coords, sparse = T)
poly_id <- which(unlist(lapply(winter_intersect, function(x) length(x) > 0)))
comyel_winter_ecoregions <- comyel_winter[poly_id,]
```

Now on to creating the migratory network model

Chapter 4

Migratory Network Model

4.1 Specifying relative abundance

To create the network model, we will first specify the relative abundance for each of the nodes (breeding and wintering). This is used to determine the overall importance of the node to the network model. In this example, we will use summarize relative abundance from the seasonal `ebirdst` rasters we created, but other data could be used as well (for example, habitat suitability from ecological niche models).

For the breeding nodes, we will input the breeding range abundance raster, genoscape polygons, and genoscape cluster names into the `MuSpTest` function `get_abundance_stats()`. The output will be a data frame of the summarized node suitability.

```
abd_season <- terra::rast("./comyel/comyel.abd_season.tif")

## Breeding
cluster_names <- genoscape_polygon_sf$Cluster
genoscape_polygon_sf <- sf::st_read("./comyel/genoscape/polygons/comyel_genoscape_polygon_sf.shp"
breeding_abundance_sum <- get_abundance_stats(range_raster = abd_season[["breeding"]],
                                              node_poly = genoscape_polygon_sf,
                                              group_names = cluster_names)
```

Similarly for the wintering nodes, we will input the nonbreeding range abundance raster, winter ecoregions, and ecoregion names into the `MuSpTest` function `get_abundance_stats()`.

```
## Nonbreeding
comyel_winter_ecoregions <- sf::st_read("./comyel/winter_regions/comyel_winter_ecoregi
ecoregion_names <- comyel_winter_ecoregions$Region
nonbreeding_abundance_sum <- get_abundance_stats(range_raster = abd_season[["nonbreedi
                                                 node_poly = comyel_winter_ecoregions,
                                                 group_names = ecoregion_names)
```

4.2 Preparing model data

The input data we need for the migratory network model are:

- Assignment matrix
- Breeding node relative abundance
- Wintering node relative abundance

In the Common Yellowthroat example, we have sampling data for wintering individuals and the assigned breeding population of those individuals. To create the assignment matrix of wintering nodes to breeding nodes, we need to determine which wintering nodes (i.e. ecoregions in this example) the Common Yellowthroat individuals are from.

4.2.1 Intersecting sampling point data with wintering nodes

In the code below, the coordinates of the wintering data are used to intersect with the wintering ecoregions the Common Yellowthroat are found in. This creates the assignment matrix `conn.df`

```
winter.points <- MuSpTest::comyel_wintering_data %>%
  sf::st_as_sf(coords=c("Long","Lat"))%>%
  sf::st_cast("MULTIPOINT")%>%
  sf::st_set_crs(4326)

conn.df <- sf::st_intersection(winter.points, comyel_winter_ecoregions) %>%
  as.data.frame() %>%
  dplyr::select(BreedingAssignment, Region) %>%
  rename("Bnode" = "BreedingAssignment",
         "Wnode" = "Region") %>%
  group_by(Bnode, Wnode) %>%
  summarize(N = n(),
            .groups = "drop") %>%
  drop_na() %>%
```

```
  pivot_wider(names_from = Wnode, values_from = N) %>%
  replace(is.na(.), 0) %>%
  column_to_rownames("Bnode")
```

4.2.2 Refining relative abundance metrics

We will do some additional refinement of the ebirdst relative abundance calculated earlier on this page in order to have it formatted properly for the migratory network model.

```
# total.mn.abd.br <- read_csv("./comyel/total.mn.abd.br.csv")
# total.mn.abd.nbr <- read_csv("./comyel/total.mn.abd.nbr.csv")
names(total.mn.abd.br) <- c("ID", "mean", "sd", "min", "max", "total")
names(total.mn.abd.nbr) <- names(total.mn.abd.br)
hsm.metrics <- rbind(total.mn.abd.br, total.mn.abd.nbr) %>%
  dplyr::mutate(area = total / mean) %>%
  dplyr::mutate(sdsummed = sd * sqrt(area))

hsm.b.total <- hsm.metrics[hsm.metrics$ID %in% rownames(conn.df), "total"]
hsm.w.total <- hsm.metrics[hsm.metrics$ID %in% colnames(conn.df), "total"]
pop.b <- round(1000*hsm.b.total/sum(hsm.b.total),0)
pop.w <- round(1000*hsm.w.total/sum(hsm.w.total),0)
```

4.3 JAGS model

4.3.1 Install JAGS and jagsUI package

The migratory network model will be run using R packages that run JAGS (Just Another Gibbs Sampler). JAGS is a specific software for conducting analysis of Bayesian hierarchical models using Markov Chain Monte Carlo simulation. JAGS can be downloaded here. Once JAGS is installed, download the `jagsUI` R package which can be found on CRAN - `install.packages("jagsUI")`.

4.3.2 Data input

The code below creates the `jags.data` object which will serve as input for the model.

```
obs_bnode_n <- nrow(conn.df)
obs_wnode_n <- ncol(conn.df)
dta_conn <- as.matrix(conn.df)
```

```
jags.data <- list(obs_bnode_n = obs_bnode_n,
                  obs_wnode_n = obs_wnode_n,
                  dta_conn_x = dta_conn,
                  dta_conn_y = dta_conn,
                  dta_conn_colsum = colSums(dta_conn),
                  dta_conn_rowsum = rowSums(dta_conn),
                  dta_conn_effort = colSums(dta_conn)/sum(dta_conn),
                  dta_conn_nb.est.mn = pop.b,
                  dta_conn_nb.est.sum = sum(pop.b),
                  dta_conn_nw.est.mn = pop.w,
                  dta_conn_nw.est.sum = sum(pop.w))
```

4.3.3 Running the model

In the code below, we will set the simulation parameters and run the model using the input data we created above. The hierarchical model is defined by the `inpm_conn_02.txt` file that is provided in this package and called below with the `system.file()` function in R. There are lots of resources if you would like to learn more about JAGS and using it in R - you can find some here and here.

```
parameters <- c("conn_g")
# parameters from prod and surv models
 ni <- 500000
 nt <- 4
 nb <- 100000
# na <- 50000  #not using
 nc <- 3

# use autojags, run until converged
model.file <- system.file("extdata", "inpm_conn_02.txt", package = "MuSpTest")
jags.out <- autojags(jags.data, inits = NULL, parameters, model.file,
                 n.chains = nc, n.thin = nt, iter.increment = ni,
                 max.iter = ni*50+nb, n.burnin = nb,
                 n.adapt= NULL, parallel=TRUE)

desc.out <- list(bnodes = rownames(conn.df),
                 wnodes = colnames(conn.df))

save(jags.out, desc.out, jags.data,
     file = "./comyel/jags_models/comyel.inpm_conn2.Rdata")
```

Voila. A migratory network model is completed. Now, onto visualization

Chapter 5

Network visualization

To be created...