Supplementary material for *Using citizen science to parse climatic and landcover influences on bird occupancy within a tropical biodiversity hotspot*

Vijay Ramesh   Pratik R. Gupte   Morgan W. Tingley   VV Robin   Ruth DeFries

2020-12-26

Contents

1 Introduction

This is supplementary material for a manuscript that uses citizen science data to model the occupancy of birds in the southern Western Ghats, India.

The main project can be found here: https://github.com/pratikunterwegs/eBirdOccupancy.

1.1 Attribution

Please contact the following in case of interest in the project.

- Vijay Ramesh (lead author)
  - PhD student, Columbia University
- Pratik Gupte (repo maintainer)
  - PhD student, University of Groningen

2 Selecting species of interest

This script shows the proportion of checklists that report a particular species across every 25km by 25km grid across the Nilgiris and the Anamalais. Using this analysis, we arrived at a final list of species for occupancy modeling.

We derived this list from inclusion criteria adapted from the State of India's Birds 2020 (Viswanathan et al. 2020). Initially, we considered all 561 species in eBird that occurred within the outlines of our study area. We then considered only those species that had a minimum of 1000 detections each between 2013 and 2019 (reducing to 303 species). Next, the study area was divided into 25 x 25 km cells following Viswanathan et al. (2020). We then kept only those species that occurred in at least 5% of all checklists across 50% of the 25 x 25 km cells from where they have been reported (reducing to 93 species). We used the above criteria to ensure as much uniform sampling of a species as possible across our study area and to reduce any erroneous associations between environmental drivers and species occupancy. Across our final list of 93 species, we analyzed a total of ~3.2 million detections (presences) between 2013 and 2019.

2.1 Prepare libraries

```
# load libraries
library(data.table)
library(readxl)
library(magrittr)
library(stringr)
library(dplyr)
library(tidyr)
library(readr)

library(ggplot2)
library(ggthemes)
library(scico)

# round any function
round_any <- function(x, accuracy = 25000) {
  round(x / accuracy) * accuracy
}
```

2.2 Read species of interest

We initally considered all species that

```
# add species of interest
specieslist <- read.csv(”data/01_list-all-spp-byCount.csv”)
speciesOfInterest <- specieslist$scientific_name
```

2.3 Load raw data for locations

```
# read in shapefile of the study area to subset by bounding box
library(sf)
wg <- st_read(”data/spatial/hillsShapefile/Nil_Ana_Pal.shp”)
box <- st_bbox(wg)

# read in data and subset
# To access the latest dataset, please visit: https://ebird.org/data/download and set the file path accordingly.
ebd <- fread(”data/ebd_IN_relApr-2020.txt”)
ebd <- ebd[between(LONGITUDE, box[”xmin”], box[”xmax”]) &
  between(LATITUDE, box[”ymin”], box[”ymax”]), ]
ebd <- ebd[year(`OBSERVATION DATE`) >= 2013, ]

# make new column names
newNames <- str_replace_all(colnames(ebd), ” ”, ”_”) %>%
  str_to_lower()
setnames(ebd, newNames)

# keep useful columns
columnsOfInterest <- c(
  ”scientific_name”, ”observation_count”, ”locality”,
  ”locality_id”, ”locality_type”, ”latitude”,
  ”longitude”, ”observation_date”, ”sampling_event_identifier”
)

ebd <- ebd[, ..columnsOfInterest]
```

Add a spatial filter and assign grids of 25km x 25km.

```r
# strict spatial filter and assign grid
locs <- ebd[, .(longitude, latitude)]

# transform to UTM and get 20km boxes
coords <- setDF(locs) %>%
  st_as_sf(coords = c("longitude", "latitude")) %>%
  `st_crs<-`(4326) %>%
  bind_cols(as.data.table(st_coordinates(.))) %>%
  st_transform(32643) %>%
  mutate(id = 1:nrow(.))

# convert wg to UTM for filter
wg <- st_transform(wg, 32643)
coords <- coords %>%
  filter(id %in% unlist(st_contains(wg, coords))) %>%
  rename(longitude = X, latitude = Y) %>%
  bind_cols(as.data.table(st_coordinates(.))) %>%
  st_drop_geometry() %>%
  as.data.table()

# remove unneeded objects
rm(locs)
gc()

coords <- coords[, .N, by = .(longitude, latitude, X, Y)]

ebd <- merge(ebd, coords, all = FALSE, by = c("longitude", "latitude"))

ebd <- ebd[(longitude %in% coords$longitude) &
  (latitude %in% coords$latitude), ]
```

2.4 Get proportional obs counts in 25km cells

```r
# round to 25km cell in UTM coords
ebd[, `:=`(X = round_any(X), Y = round_any(Y))]

# count checklists in cell
ebd_summary <- ebd[, nchk := length(unique(sampling_event_identifier)),
  by = .(X, Y)
]

# count checklists reporting each species in cell and get proportion
ebd_summary <- ebd_summary[, .(nrep = length(unique(
  sampling_event_identifier
))),
by = .(X, Y, nchk, scientific_name)
]

ebd_summary[, p_rep := nrep / nchk]

# filter for soi
ebd_summary <- ebd_summary[scientific_name %in% speciesOfInterest, ]

```

```r
# complete the dataframe for no reports
# keep no reports as NA --- allows filtering based on proportion reporting
ebd_summary <- setDF(ebd_summary) %>%
  complete(
    nesting(X, Y), scientific_name # ,
    # fill = list(p_rep = 0)
  ) %>%
  filter(!is.na(p_rep))
```

2.5 Which species are reported sufficiently in checklists?

```r
# A total of 42 unique grids (of 25km by 25km) across the study area
# total number of checklists across unique grids

tot_n_chklist <- ebd_summary %>%
  distinct(X, Y, nchk)

# species-specific number of grids
spp_grids <- ebd_summary %>%
  group_by(scientific_name) %>%
  distinct(X, Y) %>%
  count(scientific_name,
    name = ”n_grids”
  )

# Write the above two results
write_csv(tot_n_chklist, ”data/nchk_per_grid.csv”)
write_csv(spp_grids, ”data/ngrids_per_spp.csv”)

# left-join the datasets
ebd_summary <- left_join(ebd_summary, spp_grids, by = ”scientific_name”)

# check the proportion of grids across which this cut-off is met for each species
# Is it > 90% or 70%?
# For example, with a 3% cut-off, ~100 species are occurring in >50%
# of the grids they have been reported in

p_cutoff <- 0.05 # Proportion of checklists a species has been reported in
grid_proportions <- ebd_summary %>%
  group_by(scientific_name) %>%
  tally(p_rep >= p_cutoff) %>%
  mutate(prop_grids_cut = n / (spp_grids$n_grids)) %>%
  arrange(desc(prop_grids_cut))

grid_prop_cut <- filter(
  grid_proportions,
  prop_grids_cut > p_cutoff
)

# Write the results
write_csv(grid_prop_cut, ”data/chk_5_percent.csv”)

# Identifying the number of species that occur in potentially <5% of all lists
total_number_lists <- sum(tot_n_chklist$nchk)
```

```
spp_sum_chk <- ebd_summary %>%
  distinct(X, Y, scientific_name, nrep) %>%
  group_by(scientific_name) %>%
  mutate(sum_chk = sum(nrep)) %>%
  distinct(scientific_name, sum_chk)

# Approximately 90 to 100 species occur in >5% of all checklists
prop_all_lists <- spp_sum_chk %>%
  mutate(prop_lists = sum_chk / total_number_lists) %>%
  arrange(desc(prop_lists))
```

2.6 Figure: Checklist distribution

```
# add land
library(rnaturalearth)
land <- ne_countries(
  scale = 50, type = ”countries”, continent = ”asia”,
  country = ”india”,
  returnclass = c(”sf”)
)

# crop land
land <- st_transform(land, 32643)
```

2.7 Prepare the species list

```
# write the new list of species that occur in at least 5% of checklists across a minimum of 50% of the grids they h

new_sp_list <- semi_join(specieslist, grid_prop_cut, by = ”scientific_name”)

write_csv(new_sp_list, ”data/03_list-of-species-cutoff.csv”)
```

3 Landcover classification

This script was used to classify a 2019 Sentinel composite image across the Nilgiris and the Anamalais into seven distinct land cover types. The same code can be viewed on GEE here: https://code.earthengine.google.com/ec69fc4ffad32a532b25202009243d42. We use ground truthed points from a previous study (Arasumani et al. 2019).

```
// Data: Groundtruthed points from Arasumani et al 2019

// Function to obtain a Cloud-Free Image //

/**
 * Function to mask clouds using the Sentinel-2 QA band
 * @param {ee.Image} image Sentinel-2 image
 * @return {ee.Image} cloud masked Sentinel-2 image
 */

function maskS2clouds(image) {
  var qa = image.select('QA60');

  // Bits 10 and 11 are clouds and cirrus, respectively.
  var cloudBitMask = 1 << 10;
```

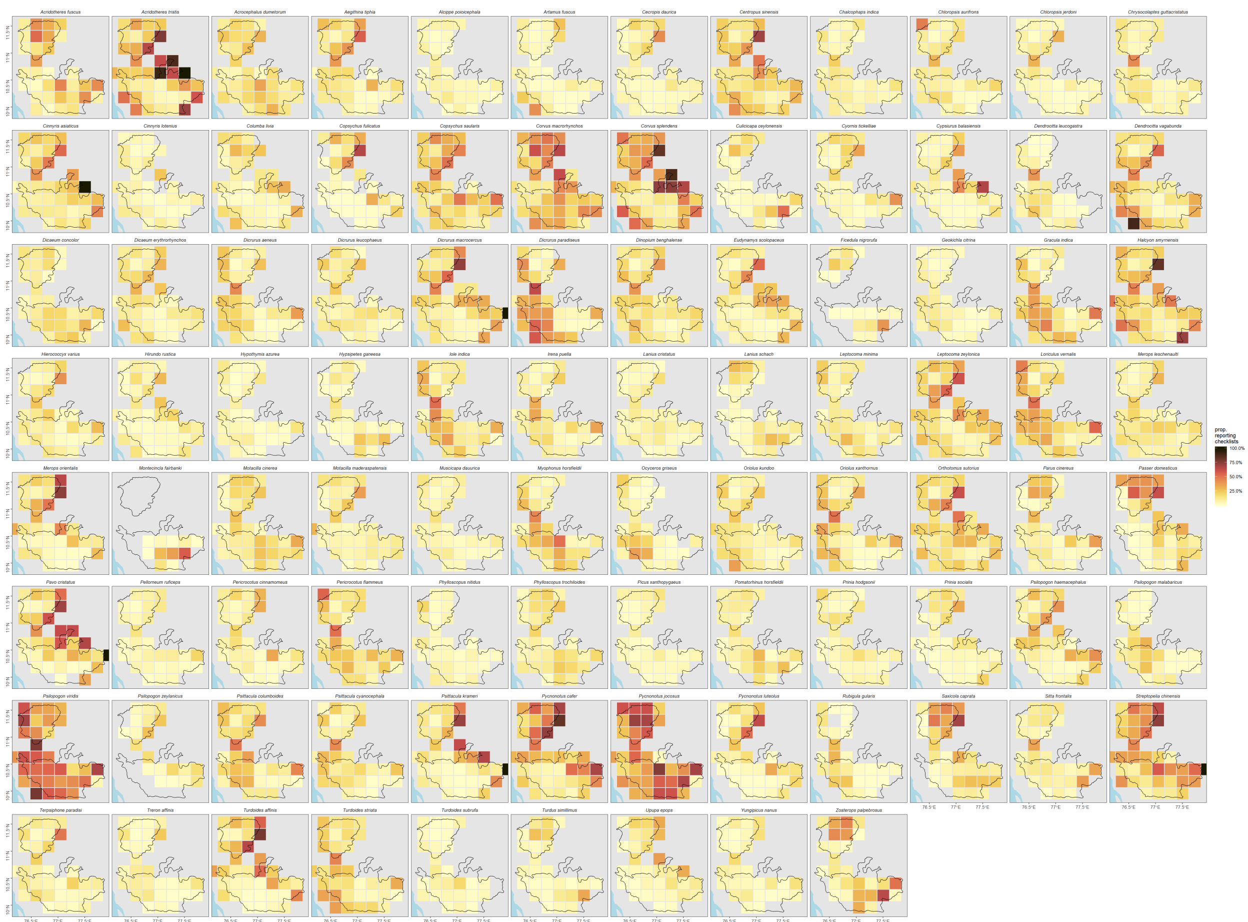

Figure 1: Proportion of checklists reporting a species in each grid cell (25km side) between 2013 and 2019. Checklists were filtered to be within the boundaries of the Nilgiris and the Anamalai hills (black outline), but rounding to 25km cells may place cells outside the boundary. Deeper shades of red indicate a higher proportion of checklists reporting a species.

```
  var cirrusBitMask = 1 << 11;

  // Both flags should be set to zero, indicating clear conditions.
  var mask = qa.bitwiseAnd(cloudBitMask).eq(0)
      .and(qa.bitwiseAnd(cirrusBitMask).eq(0));

  return image.updateMask(mask).divide(10000);
}

// Importing shapefile needed for classification
var clipper = function(image){
  return image.clip(WG_Buffer);
};


// Import raw Sentinel scenes and clip them over your study area
var filtered = sentinel.filterDate('2018-01-01','2018-12-01').map(clipper);

// Load Sentinel-2 TOA reflectance data.
// Pre-filter to get less cloudy granules.

var dataset = filtered.filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
                  .map(maskS2clouds);
var scene = dataset.reduce(ee.Reducer.median());

Map.addLayer(WG_Buffer, {}, 'Buffer Outline for Nil/Ana/Pal');
// Map.addLayer(scene,{},'Image for Classification');
// Map.addLayer(WG, {},'Outline for Nilgiris/Anaimalais/Palanis');


// Step 2: Creating training data manually
// Added a new shapefile field manually in ArcMap so that GEE can take a float field for classification
// Field: landcover
// Values: agriculture (1), forest (2), grassland (3), plantation (4), settlements (5), tea (6), waterbodies (7)
// Note - Arasu has classified plantation as Acacia, Pine et al sub classes (for future analysis)

// Merging the featureCollections to obtain a single featureCollection

var trainingFeatures = agriculture.merge(forests).merge(forests2).merge(grasslands).merge(grasslands2)
                                 .merge(settlements).merge(plantations)
                              .merge(waterbodies).merge(tea).merge(tea2).merge(tea3).merge(forests3);

// // Specify the bands of the sentinel image to be used as predictors (p)
var predictionBands = ['B2_median','B3_median','B4_median','B8_median'];


// // Now a random forest is a collection of random trees. It's predictions are used to compute an
// // average (regression) or vote on a label (classification)

var sample = scene.select(predictionBands)
                      .sampleRegions({
                        collection: trainingFeatures,
                        properties : ['landcover'],
                        scale: 10
```

```
                                });

// Let's run a classifier for randomForest
var classifier = ee.Classifier.randomForest(10).train({
                             features: sample,
                             classProperty: 'landcover',
                             inputProperties: predictionBands
});


var classified = scene.select(predictionBands).classify(classifier);
Map.addLayer(classified, {min:1, max:7,palette:[
  'be4fc4', // agriculture, violetish
  '04a310', // forests, lighter green
  'cbb315', // grasslands, yellowish
  'c17111', // plantations, brownish
  'b0a69d', // settlements, grayish
  '025a05', // tea, dark greenish
  '2035df', // waterbodies, royal blue
  ]}, 'classified');

// Partitioning training data to run an accuracy assessment
// Adding a randomColumn of values ranging from 0 to 1
var trainingTesting = sample.randomColumn();

var trainingSet = trainingTesting.filter(ee.Filter.lt('random',0.8));
var testingSet = trainingTesting.filter(ee.Filter.gte('random',0.2));

// Now run the classifier only with the trainingSet
var trained = ee.Classifier.randomForest(10).train({
  features: trainingSet,
  classProperty: 'landcover',
  inputProperties: predictionBands
});

// Now classify the testData and obtain a Confusion matrix
var confusionMatrix = ee.ConfusionMatrix(testingSet.classify(trained)
                                                    .errorMatrix({
                                                      actual: 'landcover',
                                                      predicted: 'classification'
                                                    }));

// Now print the ConfusionMatrix and expand the object to inspect the matrix()
// The entries represent the number of pixels and the items on the diagonal represent
// correct classification. Items off the diagonal are misclassifications, where class in row i
// is classified as column j

// One can also obtain basic descriptive statistics from the confusionMatrix
// Note this won't work as the number of pixels is too high (Export as .csv to obtain result)

// print('Confusion matrix:', confusionMatrix);
// print('Overall Accuracy:', confusionMatrix.accuracy());
// print('Producers Accuracy:', confusionMatrix.producersAccuracy());
// print('Consumers Accuracy:', confusionMatrix.consumersAccuracy());
```

```
// Since printing the above is gives you a computation timed out error
var exportconfusionMatrix = ee.Feature(null, {matrix: confusionMatrix.array()});
var exportAccuracy = ee.Feature(null, {matrix: confusionMatrix.accuracy()});

Export.table.toDrive({
  collection: ee.FeatureCollection(exportconfusionMatrix),
  description: 'confusionMatrix',
  fileFormat: 'CSV'
});

Export.table.toDrive({
  collection: ee.FeatureCollection(exportAccuracy),
  description: 'Accuracy',
  fileFormat: 'CSV'
});

// Below code suggests that the current projection system is WGS84
// print(classified.projection());

// To project it to UTM
var reprojected = classified.reproject('EPSG:32643',null,10);

// Export classified image
Export.image.toDrive({
  image: classified,
  description: 'Classified Image',
  scale: 10,
  region: WG_Buffer,      //.geometry().bounds(),
  fileFormat: 'GeoTIFF',
  formatOptions: {
    cloudOptimized: true
  },
  maxPixels: 618539476
});

// Export projected image
Export.image.toDrive({
  image: reprojected,
  description: 'Reprojected Image',
  scale: 10,
  region: WG_Buffer,         //.geometry().bounds(),
  fileFormat: 'GeoTIFF',
  formatOptions: {
    cloudOptimized: true
  },
  maxPixels: 618539476
});
```

4 Spatial Autocorrelation of Climatic Predictors

4.1 Load libraries

```r
# load libs
library(raster)
library(gstat)
library(stars)
library(purrr)
library(tibble)
library(dplyr)
library(tidyr)
library(glue)
library(scales)
library(gdalUtils)
library(sf)

# plot libs
library(ggplot2)
library(ggthemes)
library(scico)
library(gridExtra)
library(cowplot)
library(ggspatial)

#' make custom functiont to convert matrix to df
raster_to_df <- function(inp) {

  # assert is a raster obj
  assertthat::assert_that("RasterLayer" %in% class(inp),
    msg = "input is not a raster"
  )

  coords <- coordinates(inp)
  vals <- getValues(inp)

  data <- tibble(x = coords[, 1], y = coords[, 2], value = vals)

  return(data)
}
```

4.2 Prepare data

```r
# list landscape covariate stacks
landscape_files <- "data/spatial/landscape_resamp01km.tif"
landscape_data <- stack(landscape_files)

# get proper names
elev_names <- c("elev", "slope", "aspect")
chelsa_names <- c("bio_01", "bio_12")
names(landscape_data) <- c(elev_names, chelsa_names, "landcover")

# get chelsa rasters
chelsa <- landscape_data[[chelsa_names]]
chelsa <- purrr::map(as.list(chelsa), raster_to_df)
```

4.3 Calculate variograms of environmental layers

```
# prep variograms
vgrams <- purrr::map(chelsa, function(z) {
  z <- drop_na(z)
  vgram <- gstat::variogram(value ~ 1, loc = ~ x + y, data = z)
  return(vgram)
})

# save temp
save(vgrams, file = ”data/chelsa/chelsaVariograms.rdata”)

# get variogram data
vgrams <- purrr::map(vgrams, function(df) {
  df %>% select(dist, gamma)
})
vgrams <- tibble(
  variable = chelsa_names,
  data = vgrams
)
```

```
wg <- st_read(”data/spatial/hillsShapefile/Nil_Ana_Pal.shp”) %>%
  st_transform(32643)
bbox <- st_bbox(wg)

# Plot
library(rnaturalearth)
land <- ne_countries(
  scale = 50, type = ”countries”, continent = ”asia”,
  country = ”india”,
  returnclass = c(”sf”)
)

# crop land
land <- st_transform(land, 32643)
```

4.4 Visualise variograms of environmental data

```
# make ggplot of variograms
yaxis <- c(”semivariance”, ””)
xaxis <- c(””, ”distance (km)”)
fig_vgrams <- purrr::pmap(list(vgrams$data, yaxis, xaxis), function(df, ya, xa) {
  ggplot(df) +
    geom_line(aes(x = dist / 1000, y = gamma), size = 0.2, col = ”grey”) +
    geom_point(aes(x = dist / 1000, y = gamma), col = ”black”) +
    scale_x_continuous(labels = comma, breaks = c(seq(0, 100, 25))) +
    scale_y_log10(labels = comma) +
    labs(x = xa, y = ya) +
    theme_few() +
    theme(
      axis.text.y = element_text(angle = 90, hjust = 0.5, size = 8),
      strip.text = element_blank()
    )
})
# fig_vgrams <- purrr::map(fig_vgrams, ggplot2::ggplotGrob)
```

```
# make ggplot of chelsa data
chelsa <- as.list(landscape_data[[chelsa_names]]) %>%
  purrr::map(stars::st_as_stars)

# colour palettes
pal <- c(”bilbao”, ”davos”)
title <- c(
  ”a Annual Mean Temperature”,
  ”b Annual Precipitation”
)
direction <- c(1, 1)
lims <- list(
  range(values(landscape_data$bio_01), na.rm = T),
  range(values(landscape_data$bio_12), na.rm = T)
)
fig_list_chelsa <-
  purrr::pmap(
    list(chelsa, pal, title, direction, lims),
    function(df, pal, t, d, l) {
      ggplot() +
        stars::geom_stars(data = df) +
        geom_sf(data = land, fill = NA, colour = ”black”) +
        geom_sf(data = wg, fill = NA, colour = ”black”, size = 0.3) +
        scale_fill_scico(
          palette = pal, direction = d,
          label = comma, na.value = NA, limits = l
        ) +
        coord_sf(
          xlim = bbox[c(”xmin”, ”xmax”)],
          ylim = bbox[c(”ymin”, ”ymax”)]
        ) +
        ggspatial::annotation_scale(location = ”tr”, width_hint = 0.4, text_cex = 1) +
        theme_few() +
        theme(
          legend.position = ”top”,
          title = element_text(face = ”bold”, size = 8),
          legend.key.height = unit(0.2, ”cm”),
          legend.key.width = unit(1, ”cm”),
          legend.text = element_text(size = 8),
          axis.title = element_blank(),
          axis.text.y = element_text(angle = 90, hjust = 0.5),
          panel.background = element_rect(fill = ”lightblue”),
          legend.title = element_blank()
        ) +
        labs(x = NULL, y = NULL, title = t)
    }
  )
# fig_list_chelsa <- purrr::map(fig_list_chelsa, ggplotGrob)
```

5 Climatic raster resampling

5.1 Prepare landcover

To access the classified Sentinel image, please visit: https://code.earthengine.google.com/ec69fc4ffad32a532b25202009243d42

```
# read in landcover raster location
landcover <- ”data/landUseClassification/classifiedImage-UTM.tif”
# get extent
e <- bbox(raster(landcover))

# init resolution
res_init <- res(raster(landcover))
# res to transform to 1000m
res_final <- map(c(100, 250, 500, 1e3, 2.5e3), function(x) {
  x * res_init
})

# use gdalutils gdalwarp for resampling transform
# to 1km from 10m
for (i in 1:length(res_final)) {
  this_res <- res_final[[i]]
  this_res_char <- stringr::str_pad(this_res[1], 5, pad = ”0”)
  gdalUtils::gdalwarp(
    srcfile = landcover,
    dstfile = as.character(glue(”data/landUseClassification/lc_{this_res_char}m.tif”)),
    tr = c(this_res), r = ”mode”, te = c(e)
  )
}
```

```
# read in resampled landcover raster files as a list
lc_files <- list.files(”data/landUseClassification/”, pattern = ”lc”, full.names = TRUE)
lc_data <- map(lc_files, raster)
```

5.2 Prepare spatial extent

```
# load hills
library(sf)
hills <- st_read(”data/spatial/hillsShapefile/Nil_Ana_Pal.shp”)
hills <- st_transform(hills, 32643)
buffer <- st_buffer(hills, 3e4) %>%
  st_transform(4326)
bbox <- st_bbox(hills)
```

5.3 Prepare CHELSA rasters

Please download the CHELSA rasters from https://chelsa-climate.org/bioclim/

```
# list chelsa files
chelsaFiles <- list.files(”data/chelsa/”, full.names = TRUE, pattern = ”*.tif”)

# gather chelsa rasters
chelsaData <- purrr::map(chelsaFiles, function(chr) {
  a <- raster(chr)
  crs(a) <- crs(buffer)
  a <- crop(a, as(buffer, ”Spatial”))
```

```
  return(a)
})

# stack chelsa data
chelsaData <- raster::stack(chelsaData)
names(chelsaData) <- c(”chelsa_bio10_01”, ”chelsa_bio10_12”)
```

5.4 Resample prepared rasters

```
# make resampled data
resamp_data <- map(lc_data, function(this_scale) {
  rr <- projectRaster(
    from = chelsaData, to = this_scale,
    crs = crs(this_scale), res = res(this_scale)
  )
})

# make a stars list
resamp_data <- map2(resamp_data, lc_data, function(z1, z2) {
  z2[z2 == 0] <- NA
  z2 <- append(z2, as.list(z1)) %>% map(stars::st_as_stars)
}) %>%
  flatten()
```

6 Climate in Relation to Landcover

This script showcases how climatic predictors vary as a function of land cover types across our study area.

6.1 Prepare libraries

```
# load libs
library(raster)
library(glue)
library(purrr)
library(dplyr)
library(tidyr)

# plotting options
library(ggplot2)
library(ggthemes)
library(scico)

# get ci func
ci <- function(x) {
  qnorm(0.975) * sd(x, na.rm = T) / sqrt(length(x))
}
```

6.2 Prepare environmental data

```
# read landscape prepare for plotting
landscape <- stack(”data/spatial/landscape_resamp01km.tif”)

# get proper names
elev_names <- c(”elev”, ”slope”, ”aspect”)
```

```
chelsa_names <- c(”bio_01”, ”bio_12”)

names(landscape) <- as.character(glue('{c(elev_names, chelsa_names, ”landcover”)}'))
```

```
# make duplicate stack
land_data <- landscape[[c(”landcover”, chelsa_names)]]

# convert to list
land_data <- as.list(land_data)

# map get values over the stack
land_data <- purrr::map(land_data, raster::getValues)
names(land_data) <- c(”landcover”, chelsa_names)

# conver to dataframe and round to 100m
land_data <- bind_cols(land_data)
land_data <- drop_na(land_data) %>%
  filter(landcover != 0) %>%
  pivot_longer(
    cols = contains(”bio”),
    names_to = ”clim_var”
  ) # %>%
# group_by(landcover, clim_var) %>%
# summarise_all(.funs = list(~mean(.), ~ci(.)))
```

6.3 Climatic variables over landcover

Figure code is hidden in versions rendered as HTML and PDF.

7 Distribution of Observer Expertise

This script plots observer expertise over time (2013-2019) as well as across land cover types.

7.1 Prepare libraries

```
# load libs
library(raster)
library(glue)
library(purrr)
library(dplyr)
library(tidyr)
library(readr)
library(scales)

# plotting libs
library(ggplot2)
library(ggthemes)
library(scico)

# get ci func
ci <- function(x) {
  qnorm(0.975) * sd(x, na.rm = T) / sqrt(length(x))
}
```

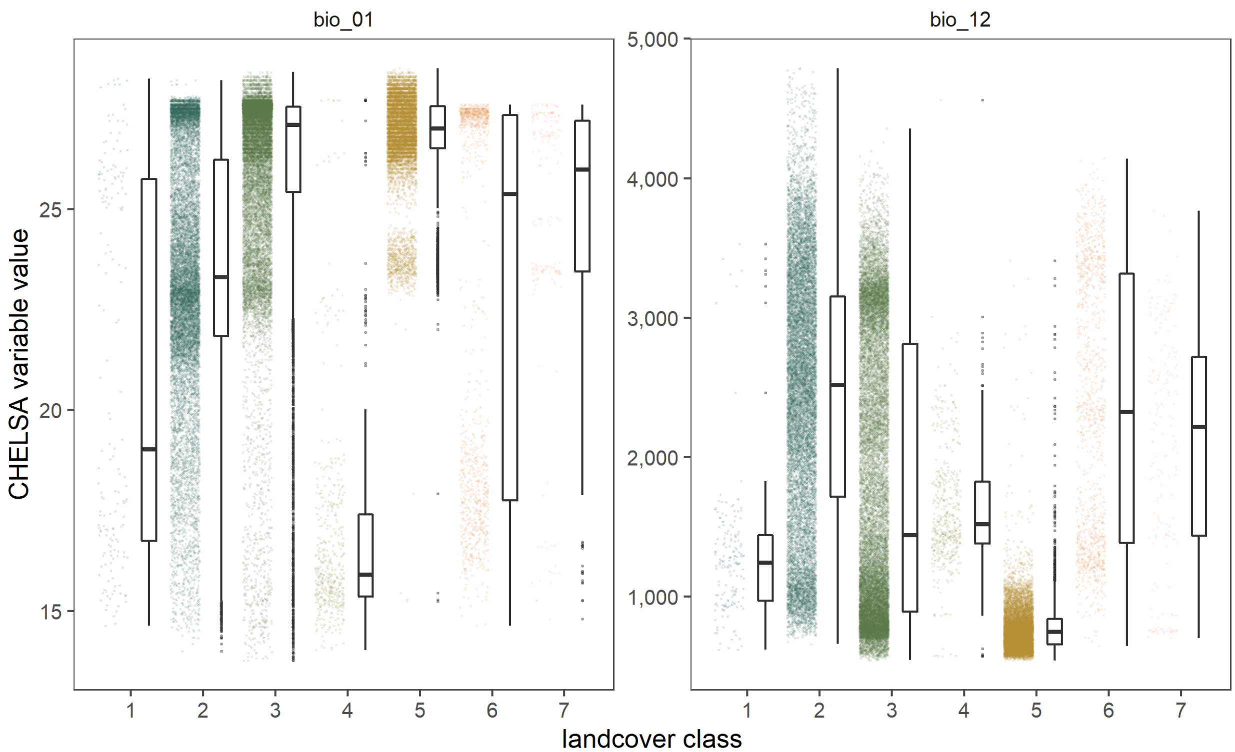

Figure 2: CHELSA climatic variables (Annual Mean Temperature on the left and Annual Precipitation on the right) are plotted as a function of landcover type. Grey points in the background represent raw data.

7.2 Load observer expertise scores and checklist covariates

```
# read in scores and checklist data and link
scores <- read_csv(”data/03_data-obsExpertise-score.csv”)
data <- read_csv(”data/03_data-covars-perChklist.csv”)

data <- left_join(data, scores, by = c(”observer” = ”observer”))
data <- dplyr::select(data, score, nSp, nSoi, landcover, year) %>%
  filter(!is.na(score))
```

7.3 Species observed in relation to observer expertise

```
# summarise data by rounded score and year
data_summary01 <- data %>%
  mutate(score = plyr::round_any(score, 0.2)) %>%
  dplyr::select(score, year, nSp, nSoi) %>%
  pivot_longer(
    cols = c(”nSp”, ”nSoi”),
    names_to = ”variable”, values_to = ”value”
  ) %>%
  group_by(score, year, variable) %>%
  summarise_at(vars(value), list(~ mean(.), ~ ci(.)))

# make plot and export
fig_nsp_score <-
  ggplot(data_summary01) +
  geom_jitter(
    data = data, aes(x = score, y = nSp),
    col = ”grey”, alpha = 0.2, size = 0.1
  ) +
  geom_pointrange(aes(
    x = score, y = mean,
    ymin = mean - ci, ymax = mean + ci,
    col = as.factor(variable)
  ),
  position = position_dodge(width = 0.05)
  ) +
  facet_wrap(~year) +
  scale_y_log10() +
  #  coord_cartesian(ylim=c(0,50))+
  scale_colour_scico_d(palette = ”cork”, begin = 0.2, end = 0.8) +
  labs(x = ”CCI”, y = ”Number of Species Reported”) +
  theme_few() +
  theme(legend.position = ”none”)

# export figure
ggsave(filename = ”figs/fig_nsp_score.png”, width = 12, height = 7, device = png(), dpi = 300)
dev.off()
```

7.4 Observer expertise in relation to landcover

Figure code is hidden in versions rendered as HTML or PDF.

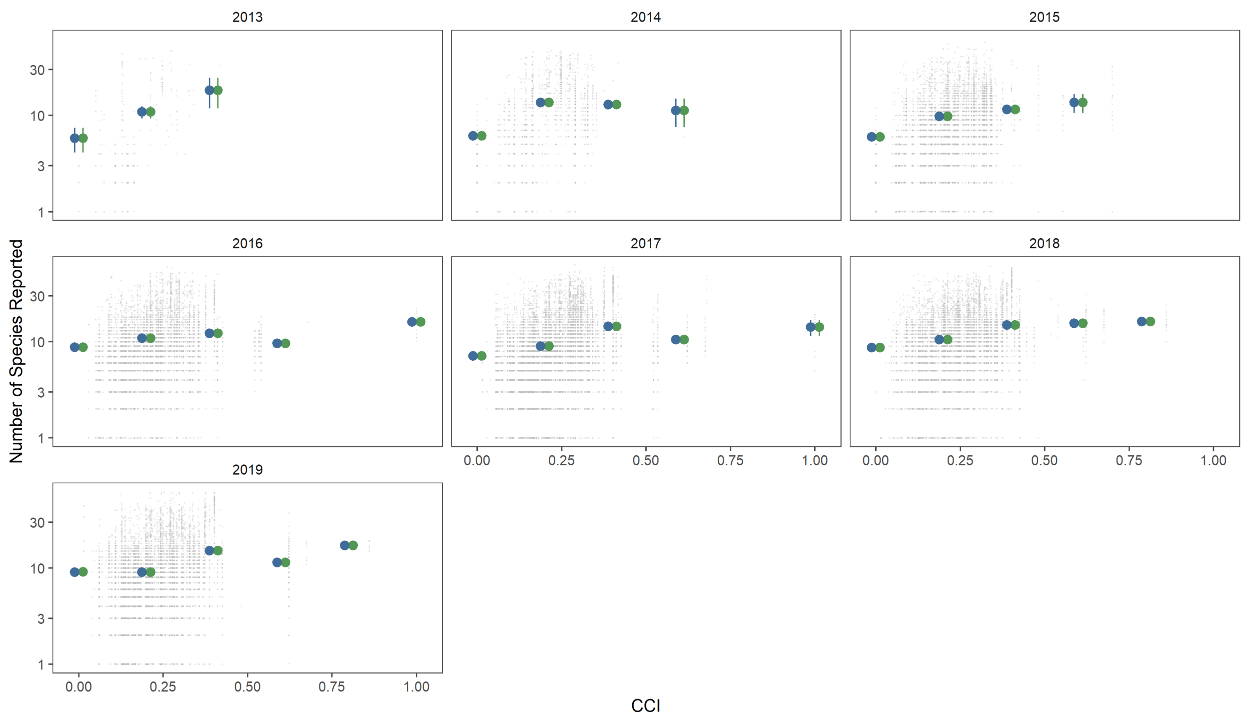

Figure 3: Total number of species (blue) and species of interest to this study (green) reported in checklists from the study area over the years 2013 – 2019, as a function of the expertise score of the reporting observer. Points represent means, with bars showing the 95% confidence intervals; data shown are for expertise scores rounded to multiples of 0.2, and the y-axis is on a log scale. Raw data are shown in the background (grey points).

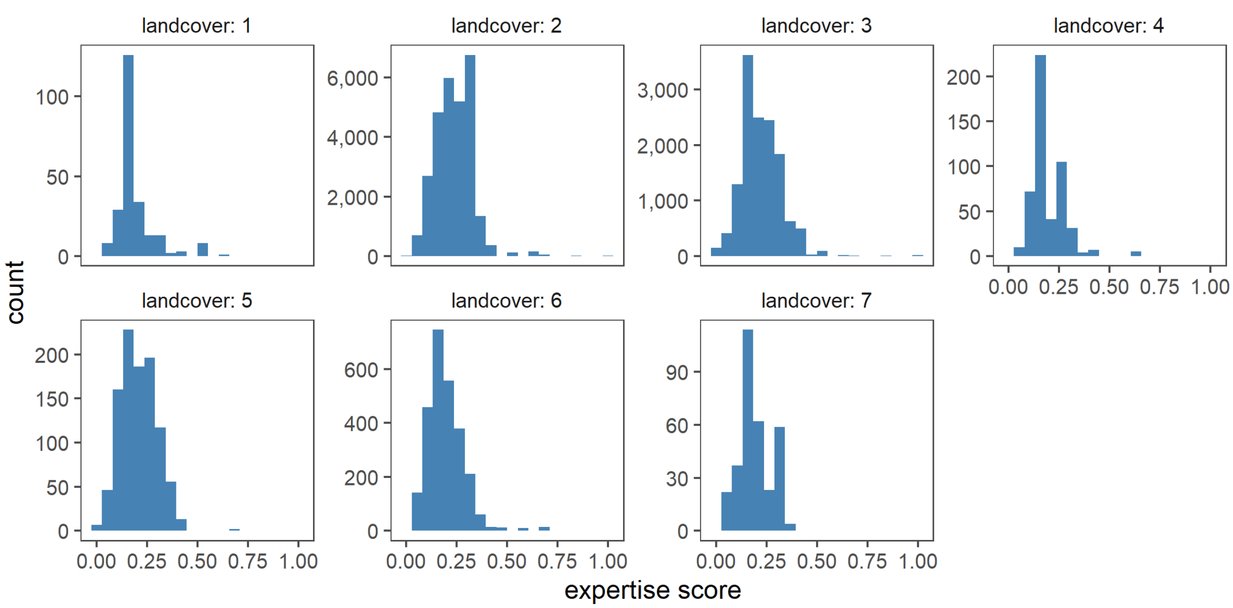

Figure 4: Distribution of expertise scores across the seven landcover classes present in the study site.

8 Matching Effort Cutoffs with Spatial Independence Criteria

How many sites would be lost if effort distance was restricted based on spatial independence?

8.1 Load librarires

```
# load data packagaes
library(data.table)
library(dplyr)

# load plotting packages
library(ggplot2)
library(scico)
library(ggthemes)
library(scales)
```

8.2 Load data

```
# load checklist covariates
data <- fread(”data/03_data-covars-perChklist.csv”)

effort_distance_summary <- data[, effort_distance_class :=
  cut(distance, breaks = c(
    -1, 0.001, 0.1, 0.25,
    0.5, 1, 2.5, 5, Inf
  ), ordered_result = T)][,
  .N,
  by = effort_distance_class
][
  order(effort_distance_class)
]

effort_distance_summary[
  ,
  prop_effort := cumsum(effort_distance_summary$N) / nrow(data)
]
```

8.3 Visualise limiting effort by spatial independence limits

```
# plot effort distance class cumulative sum
fig_dist_exclusion <- ggplot(effort_distance_summary) +
  geom_point(aes(effort_distance_class, prop_effort), size = 3) +
  geom_path(aes(effort_distance_class, prop_effort, group = NA)) +
  # scale_y_continuous(label=label_number(scale=0.001, accuracy = 1, suffix = ”K”))+
  scale_x_discrete(labels = c(
    ”stationary”, ”100m”, ”250m”,
    ”500m”, ”1 km”, ”2.5 km”, ”5 km”
  )) +
  theme_few() +
  theme(panel.grid = element_line(size = 0.2, color = ”grey”)) +
  labs(x = ”effort distance cutoff”, y = ”proportion of checklists”)

ggsave(
  plot = fig_dist_exclusion, ”figs/fig_cutoff_effort.png”,
  height = 6, width = 8, dpi = 300
```

```r
)
dev.off()
```

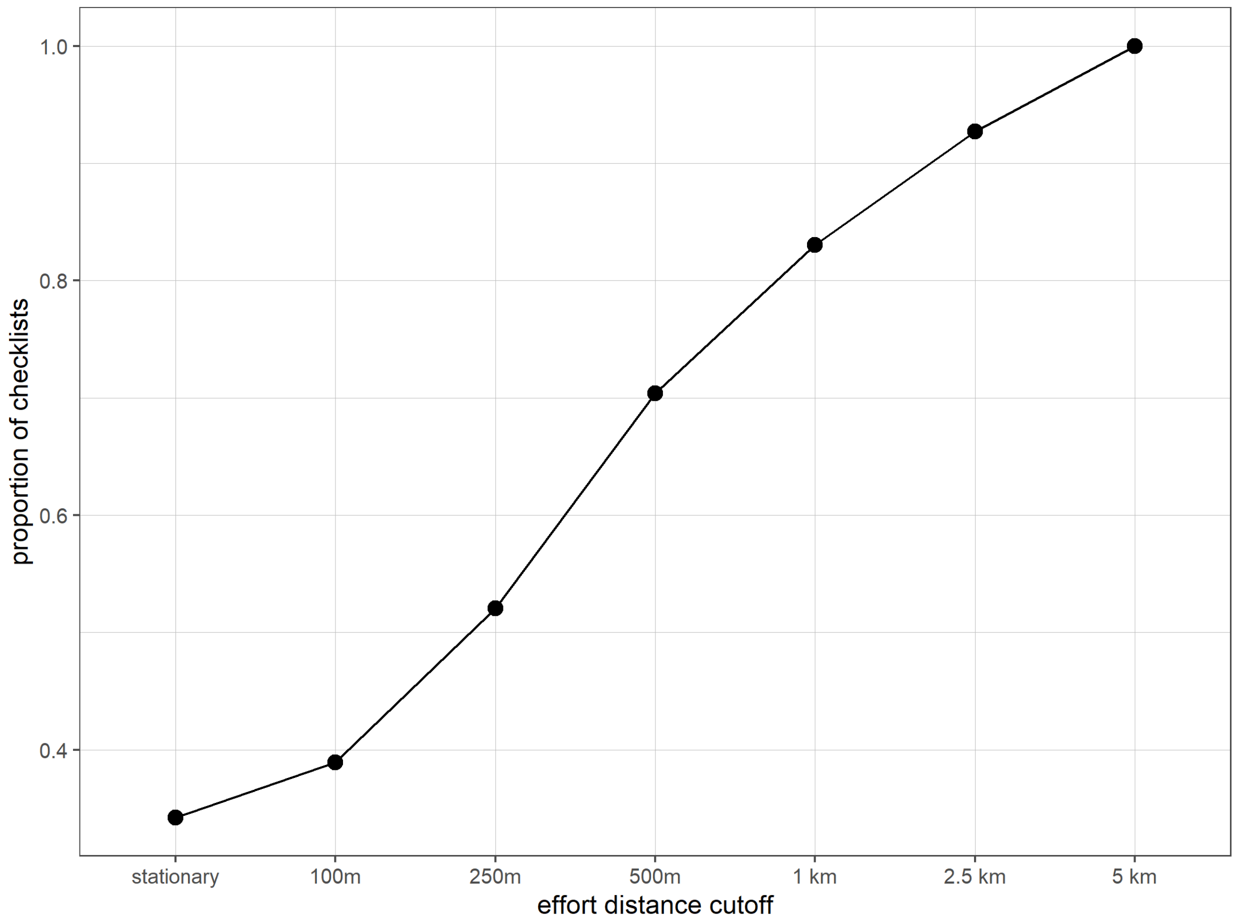

Figure 5: Proportion of checklists that are retained across the study area as a function of the distance cutoff specified. For instance, 80% of checklists are retained with a distance cutoff of 1km.

9 Spatial Thinning: A Brief Comparison of Approaches

9.1 Prepare libraries

```r
# load libraries
library(tidyverse)
library(glue)
library(readr)
library(sf)

# plotting
library(ggthemes)
library(scico)
library(scales)

# ci func
ci <- function(x) {
```

```
  qnorm(0.975) * sd(x, na.rm = T) / sqrt(length(x))
}

# load python libs here
library(reticulate)
# set python path
use_python(”/usr/bin/python3”)
```

9.2 Traditional grid-based thinning

```
# load the shapefile of the study area
wg <- st_read(”data/spatial/hillsShapefile/Nil_Ana_Pal.shp”) %>%
  st_transform(32643)

# get scales
# load checklist data and select one per rounded 500m coordinates
{
  data <- read_csv(”data/03_data-covars-perChklist.csv”) %>%
    count(longitude, latitude, name = ”tot_effort”)


  # how many unique points
  n_all_points <- nrow(data)
  d_all_effort <- sum(data$tot_effort)

  # round to different scales
  scale <- c(100, 250, 500, 1000)

  # group data by scale
  data <- crossing(scale, data) %>%
    group_by(scale) %>%
    nest() %>%
    ungroup()
}

# select one point per grid cell
data <- mutate(data, data = map2(scale, data, function(sc, df) {
  # transform the data
  df <- df %>%
    st_as_sf(coords = c(”longitude”, ”latitude”)) %>%
    `st_crs<-`(4326) %>%
    st_transform(32643) %>%
    bind_cols(as_tibble(st_coordinates(.))) %>%
    mutate(
      coordId = 1:nrow(.),
      X_round = plyr::round_any(X, sc),
      Y_round = plyr::round_any(Y, sc)
    )

  # make a grid
  grid <- st_make_grid(wg, cellsize = sc)

  # which cell contains which points
  grid_contents <- st_contains(grid, df) %>%
```

```
    as_tibble() %>%
    rename(cell = row.id, coordId = col.id)

  rm(grid)

  # what's the max point in each grid
  points_max <- left_join(df %>% st_drop_geometry(),
    grid_contents,
    by = "coordId"
  ) %>%
    group_by(cell) %>%
    filter(tot_effort == max(tot_effort))

  # get summary for max
  max_sites <- points_max %>%
    ungroup() %>%
    summarise(
      prop_points = length(coordId) / n_all_points,
      prop_effort = sum(tot_effort) / d_all_effort
    ) %>%
    pivot_longer(
      cols = everything(),
      names_to = "variable"
    )

  # select a random point in each grid
  points_rand <- left_join(df %>% st_drop_geometry(),
    grid_contents,
    by = "coordId"
  ) %>%
    group_by(cell) %>%
    sample_n(size = 1)

  # get summary for rand
  rand_sites <- points_rand %>%
    ungroup() %>%
    summarise(
      prop_points = length(coordId) / n_all_points,
      prop_effort = sum(tot_effort) / d_all_effort
    ) %>%
    pivot_longer(
      cols = everything(),
      names_to = "variable"
    )

  df <- tibble(
    grid_rand = list(rand_sites), grid_max = list(max_sites),
    points_rand = list(points_rand), points_max = list(points_max)
  )
}))

# unnest data
data <- unnest(data, cols = data)
```

```
# save summary as another object
data_thin_trad <- data %>%
  select(-contains(”points”)) %>%
  pivot_longer(
    cols = -contains(”scale”),
    names_to = ”method”, values_to = ”somedata”
  ) %>%
  unnest(cols = somedata)

# save points for later comparison
points_thin_trad <- data %>%
  select(contains(”points”), scale)

rm(data)
```

9.3 Network-based thinning

Load python libraries.

```
# import classic python libs
import numpy as np
import matplotlib.pyplot as plt

# libs for dataframes
import pandas as pd

# network lib
import networkx as nx

# import libs for geodata
import geopandas as gpd

# import ckdtree
from scipy.spatial import cKDTree
```

9.4 Finding modularity in proximity networks

```
# read in checklist covariates for conversion to gpd
# get unique coordinates, assign them to the df
# convert df to geo-df
chkCovars = pd.read_csv(”data/03_data-covars-perChklist.csv”)
ul = chkCovars[['longitude', 'latitude']].drop_duplicates(subset=['longitude', 'latitude'])
ul['coordId'] = np.arange(0, ul.shape[0])

# get effort at each coordinate
effort = chkCovars.groupby(['longitude', 'latitude']).size().to_frame('tot_effort')
effort = effort.reset_index()

# merge effort on ul
ul = pd.merge(ul, effort, on=['longitude', 'latitude'])

# make gpd and drop col from ul
ulgpd = gpd.GeoDataFrame(ul, geometry=gpd.points_from_xy(ul.longitude, ul.latitude))
ulgpd.crs = {'init' :'epsg:4326'}
# reproject spatials to 43n epsg 32643
```

```
ulgpd = ulgpd.to_crs({'init': 'epsg:32643'})
ul = pd.DataFrame(ul.drop(columns="geometry"))

# function to use ckdtrees for nearest point finding
def ckd_pairs(gdfA, dist_indep):
    A = np.concatenate([np.array(geom.coords) for geom in gdfA.geometry.to_list()])
    ckd_tree = cKDTree(A)
    dist = ckd_tree.query_pairs(r=dist_indep, output_type='ndarray')
    return dist

# define scales in metres
scales = [100, 250, 500, 1000]


# function to process ckd_pairs
def make_modules(scale):
    site_pairs = ckd_pairs(gdfA=ulgpd, dist_indep=scale)
    site_pairs = pd.DataFrame(data=site_pairs, columns=['p1', 'p2'])
    site_pairs['scale'] = scale
    # get site ids
    site_id = np.concatenate((site_pairs.p1.unique(), site_pairs.p2.unique()))
    site_id = np.unique(site_id)
    # make network
    network = nx.from_pandas_edgelist(site_pairs, 'p1', 'p2')
    # get modules
    modules = list(nx.algorithms.community.greedy_modularity_communities(network))
    # get modules as df
    m = []
    for i in np.arange(len(modules)):
        module_number = [i] * len(modules[i])
        module_coords = list(modules[i])
        m = m + list(zip(module_number, module_coords))
    # add location and summed sampling duration
    unique_locs = ul[ul.coordId.isin(site_id)]
    module_data = pd.DataFrame(m, columns=['module', 'coordId'])
    module_data = pd.merge(module_data, unique_locs, on='coordId')
    # add scale
    module_data['scale'] = scale
    return [site_pairs, module_data]


# run make modules on ulgpd at scales
data = list(map(make_modules, scales))

# extract data for output
tot_pair_data = []
tot_module_data = []
for i in np.arange(len(data)):
    tot_pair_data.append(data[i][0])
    tot_module_data.append(data[i][1])

tot_pair_data = pd.concat(tot_pair_data, ignore_index=True)
tot_module_data = pd.concat(tot_module_data, ignore_index=True)
```

```
# make dict of positions and array of coordinates
# site_id = np.concatenate((site_pairs.p1.unique(), site_pairs.p2.unique()))
# site_id = np.unique(site_id)
# locations_df = ul[ul.coordId.isin(site_id)][['longitude', 'latitude']].to_numpy()
# pos_dict = dict(zip(site_id, locations_df))

# output data
tot_module_data.to_csv(path_or_buf=”data/site_modules.csv”, index=False)
tot_pair_data.to_csv(path_or_buf=”data/site_pairs.csv”, index=False)

# ends here
```

9.5 Process proximity networks in R

```
# read in pair and module data
pairs <- read_csv(”data/site_pairs.csv”)
mods <- read_csv(”data/site_modules.csv”)

# count pairs at each scale
count(pairs, scale)
pairs %>%
  group_by(scale) %>%
  summarise(non_indep_pairs = length(unique(c(p1, p2))) / n_all_points)
count(mods, scale)

# nest by scale and add module data
data <- nest(pairs, data = c(p1, p2))
modules <- group_by(mods, scale) %>%
  nest() %>%
  ungroup()

# add module data
data <- mutate(data,
  modules = modules$data,
  data = map2(data, modules, function(df, m) {
    df <- left_join(df, m, by = c(”p1” = ”coordId”))
    df <- left_join(df, m, by = c(”p2” = ”coordId”))

    df <- filter(df, module.x == module.y)
    return(df)
  })
) %>%
  select(-modules)

# split by module
data$data <- map(data$data, function(df) {
  df <- group_by(df, module.x, module.y) %>%
    nest() %>%
    ungroup()
  return(df)
})
```

9.6 A function that removes sites

```
# a function to remove sites
remove_which_sites <- function(pair_data) {
  {
    a <- pair_data %>%
      select(p1, p2)

    nodes_a_init <- unique(c(a$p1, a$p2))

    i_n_d <- filter(mods, coordId %in% nodes_a_init) %>%
      select(node = coordId, tot_effort) %>%
      mutate(s_f_r = NA)

    nodes_keep <- c()
    nodes_removed <- c()
  }

  while (nrow(a) > 0) {

    # how many nodes in a
    nodes_a <- unique(c(a$p1, a$p2))

    # get node or site efforts and arrange in ascending order
    b <- i_n_d %>% filter(node %in% nodes_a)

    for (i in 1:nrow(b)) {
      # which node to remove
      node_out <- b$node[i]
      # how much tot_effort lost
      d_n_o <- b$tot_effort[i]

      # how many rows remain in a if node_out is removed?
      a_n_o <- filter(a, p1 != node_out, p2 != node_out)
      indep_nodes <- setdiff(nodes_a, unique(c(a_n_o$p1, a_n_o$p2, node_out)))

      # how much sampling effort made spatially independent
      indep_sampling <- filter(b, node %in% indep_nodes) %>%
        summarise(tot_effort = sum(tot_effort)) %>%
        .$tot_effort

      # message(glue::glue('{node_out} removal frees {indep_sampling} m'))
      # sampling freed by sampling lost
      b$s_f_r[i] <- indep_sampling / d_n_o
    }

    # arrange node data by decreasing sfr and increasing tot_effort
    # highest tot_effort nodes are processed last
    b <- arrange(b, -s_f_r, tot_effort)

    nodes_removed <- c(nodes_removed, b$node[1])

    # remove pairs of nodes containing the highest sfr node in b
    a <- filter(a, p1 != b$node[1], p2 != b$node[1])
```

```r

    nodes_keep <- c(nodes_keep, setdiff(nodes_a, unique(c(a$p1, a$p2, nodes_removed))))
  }

  message(glue::glue("keeping {length(nodes_keep)} of {length(nodes_a_init)}"))

  # node_status <- tibble(nodes = c(nodes_keep, nodes_removed),
  #                        status = c(rep(TRUE, length(nodes_keep)),
  #                                   rep(FALSE, length(nodes_removed))))

  return(as.integer(nodes_removed))
}
```

9.7 Removing non-independent sites

```r
# remove 5km and 2.5km scale
data <- data %>% filter(scale <= 1000)
# run select sites on the various modules
sites_removed <- map(data$data, function(df) {
  remove_sites <- unlist(purrr::map(df$data, remove_which_sites))
})

# save as rdata
save(sites_removed, file = "data/data_network_sites_removed.rdata")
```

```r
# get python sites
ul <- py$ul

load("data/data_network_sites_removed.rdata")

# subset sites
data <- mutate(data,
  data = map(sites_removed, function(site_id) {
    as_tibble(filter(ul, !coordId %in% site_id))
  })
)

# which points are kept
points_thin_net <- mutate(data,
  data = map(data, function(df) {
    df <- df %>%
      select("longitude", "latitude") %>%
      st_as_sf(coords = c("longitude", "latitude")) %>%
      `st_crs<-`(4326) %>%
      st_transform(32643) %>%
      bind_cols(as_tibble(st_coordinates(.))) %>%
      st_drop_geometry()
  })
)

# get metrics for method
data_thin_net <- unnest(data, cols = "data") %>%
  group_by(scale) %>%
  summarise(
    prop_points = length(coordId) / n_all_points,
```

```r
    prop_effort = sum(tot_effort) / d_all_effort
  ) %>%
  mutate(method = "network") %>%
  pivot_longer(
    cols = -one_of(c("method", "scale")),
    names_to = "variable"
  )
```

9.8 Measuring method fallibility

How many points, at different spatial scales, remain after the application of each method?

9.9 Prepare data for Python

```r

# get points by each method
points_list <- append(points_thin_net$data, values = append(
  points_thin_trad$points_rand,
  points_thin_trad$points_max
))

# get scales as list
scales_list <- list(100, 250, 500, 1000, rep(c(100, 250, 500, 1000), 2)) %>% flatten()

# send to python
py$points_list <- points_list
py$scales_list <- scales_list
```

9.10 Count props under threshold in Python

```python
# a function to convert to gpd
def make_gpd(df):
    df = gpd.GeoDataFrame(df, geometry=gpd.points_from_xy(df.X, df.Y))
    df.crs = {'init' :'epsg:32643'}
    return df


# function for mean nnd
# function to use ckdtrees for nearest point finding
def ckd_test(gdfA, gdfB, dist_indep):
    A = np.concatenate([np.array(geom.coords) for geom in gdfA.geometry.to_list()])
    #simplified_features = simplify_roads(gdfB)
    B = np.concatenate([np.array(geom.coords) for geom in gdfB.geometry.to_list()])
    #B = np.concatenate(B)
    ckd_tree = cKDTree(B)
    dist, idx = ckd_tree.query(A, k=[2])
    dist_diff = list(map(lambda x: x - dist_indep, dist))
    mean_dist_diff = np.asarray(dist_diff).mean()
    return mean_dist_diff


# apply to all data
points_list = list(map(make_gpd, points_list))

```

```
# get nnb all data
mean_dist_diff = list(map(ckd_test, points_list, points_list, scales_list))
```

9.11 Plot metrics for different methods

```
# combine the thinning metrics data
data_plot <- bind_rows(data_thin_net, data_thin_trad)

# get data for mean distance
data_thin_compare <- tibble(
  scale = unlist(scales_list),
  method = c(
    rep(”network”, 4),
    rep(”grid_rand”, 4),
    rep(”grid_max”, 4)
  ),
  `mean NND - buffer (m)` = unlist(py$mean_dist_diff)
) %>%
  pivot_longer(
    cols = ”mean NND - buffer (m)”,
    names_to = ”variable”
  )

# bind rows with other data
data_plot <- bind_rows(data_plot, data_thin_compare)

# plot results
fig_spatial_thinning <-
  ggplot(data_plot) +
  geom_vline(xintercept = scale, lty = 3, colour = ”grey”, lwd = 0.4) +
  geom_line(aes(x = scale, y = value, col = method)) +
  geom_point(aes(x = scale, y = value, col = method, shape = method)) +
  facet_wrap(~variable, scales = ”free”) +
  scale_shape_manual(values = c(1, 2, 0)) +
  scale_x_continuous(breaks = scale) +
  scale_y_continuous() +
  scale_colour_scico_d(palette = ”batlow”, begin = 0.2, end = 0.8) +
  theme_few() +
  theme(legend.position = ”top”) +
  labs(x = ”buffer distance (m)”)

# save
ggsave(fig_spatial_thinning,
  filename = ”figs/fig_spatial_thinning_02.png”, width = 10, height = 4,
  dpi = 300
)
dev.off()
```

10 Predicting Species-specific Occupancy

This supplement plots species-specific probabilities of occupancy as a function of significant environmental predictors.

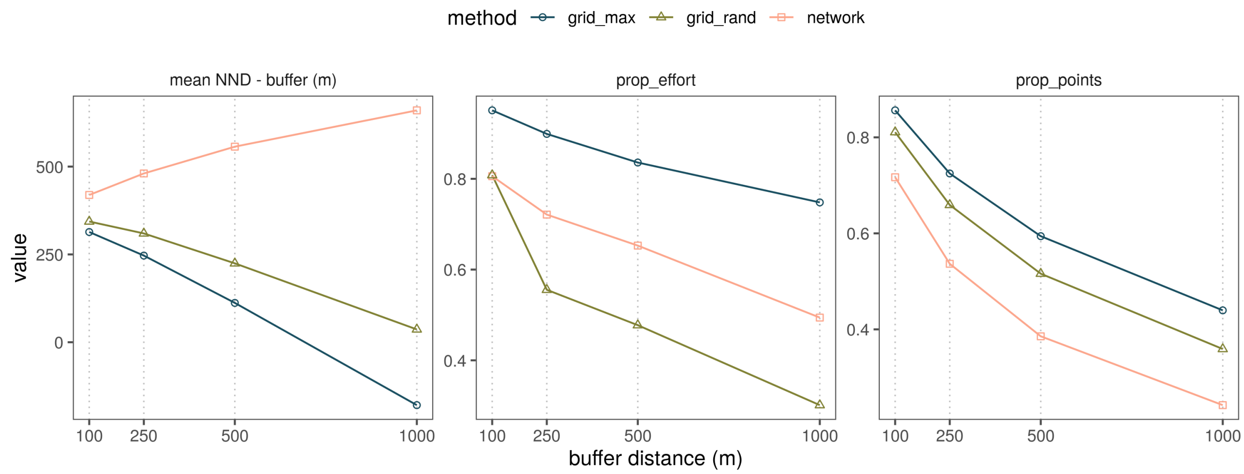

Figure 6: Three spatial thinning approaches were tested and we show that the largest proportion of effort as well as detections could be retained using a traditional grid based approach

10.1 Prepare libraries

```r
# to load data
library(readxl)

# to handle data
library(dplyr)
library(readr)
library(forcats)
library(tidyr)
library(purrr)
library(stringr)

# plotting
library(ggplot2)
library(patchwork)
```

10.2 Read data

```r
# read data
data <- read_csv("data/results/data_occupancy_predictors.csv")
```

```r
# drop na
data <- select(
  data,
  -ci
) %>%
  drop_na() %>%
  nest(data = c(predictor, m_group, seq_x, mean, scale))
```

11 References

Arasumani, M., Danish Khan, C.K. Vishnudas, M. Muthukumar, Milind Bunyan, and V.V. Robin. 2019. "Invasion Compounds an Ecosystem-Wide Loss to Afforestation in the Tropical Grasslands of the Shola Sky Islands." *Biological Conservation* 230 (February): 141–50. https://doi.org/10.1016/j.biocon.2018.12.019.

Viswanathan, A., A. Reddy, A. Deomurari, K. Suryawanshi, M. D. Madhusudan, M. Kaushik, P. J, R. Jayapal, and S. & Quader. 2020. *State of India's Birds 2020: Background and Methodology.* Manual.