Realistic topographical distance between populations of Heliconius

Problem: geographical distance not representative of real geographic isolation in habitats with complex topographies (e.g. the Andes)

Solution: obtain Topographic Least Cost Path, which includes a resistance matrix indicating the habitats a species can inhabit/move through, and models the "easiest" way for a species to move from A to B.

Tutorials/reading
https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.13317
https://rdrr.io/cran/topoDistance/f/vignettes/topoDistance-vignette.Rmd
https://cran.r-project.org/web/packages/elevatr/vignettes/introduction_to_elevatr.html

Example

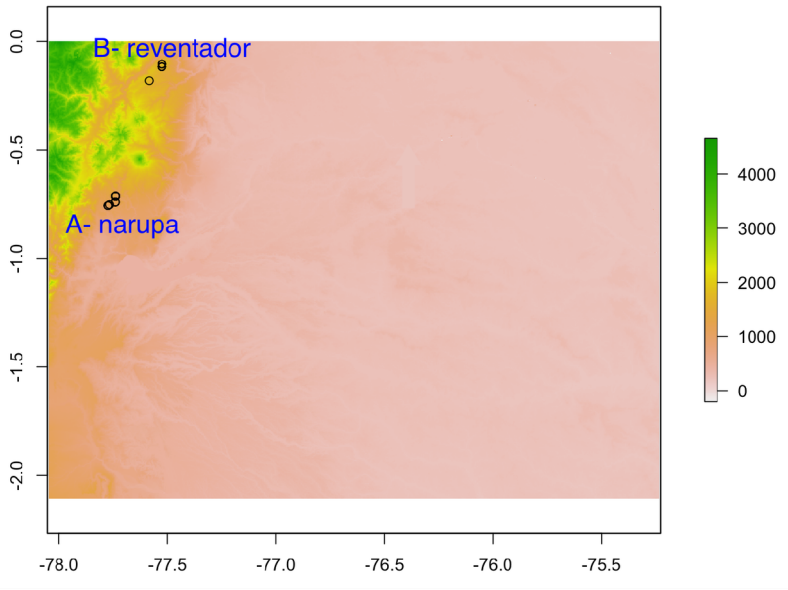
Preamble, the populations

popA= narupa reserve (nar, 1200m.a.sl, Eastern Ecuador)
popB= reventador (rev, 1500m.a.s.l, Eastern Ecuador)

There is a volcano between these two populations (Sumaco, in fact), so assuming our Heliconius butterflies can get from A to B in a straight line is far from the truth. These populations are very isolated by topography, and we ought to account for that in our models.

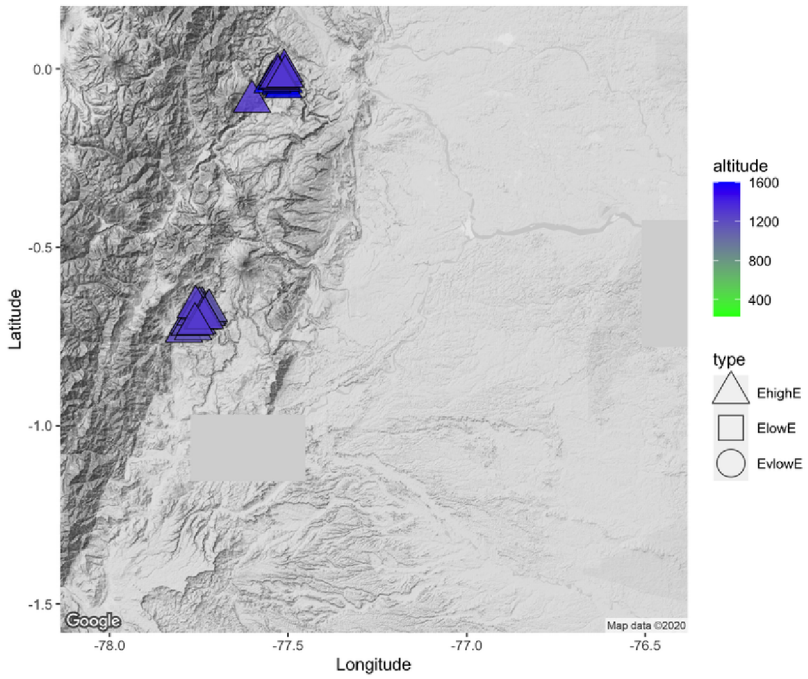
Let's look at the issue visually:

```
library(elevatr)
# prep data, mean lat/lon per population, and pop names
era.ec.e.df <- dplyr::summarise(dplyr::group_by(era.ec.e.loc, pop.name),
                                x=mean(longitude),
                                y=mean(latitude)); era.ec.e.df
# settings for elevatr
prj_dd <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
# get points into right format
era.ec.e.sp <- SpatialPoints(era.ec.e.df[,2:3], proj4string = CRS(prj_dd))
# obtain dem(digital elevation model)
era.ec.e.dem <-  get_elev_raster(era.ec.e.sp, z = 10, prj = prj_dd)
# LEFT: plot elevation map
plot(era.ec.e.dem )
points(era.ec.e.df[,2:3])
```



I like google maps contour maps too. You will have to get an API key here - if you register as an academic it's free

```
library(ggmap)
library(RgoogleMaps)
#you will have to register an API key (private)
register_google(key="XXX", write = TRUE)

# input means of coordinates to get google maps

mean(era.loc$latitude);mean(era.loc$longitude)
era.map <- get_googlemap(center=c(lon=-74.5, lat=0.5), zoom = 6, color="bw",  maptype
                         style = "feature:road|visibility:off&style=element:labels|v

era$alt.type <- str_sub(era$type, 2, 4)
gc(); p1 <- ggmap(era.map) +
  geom_jitter(data = era, colour="black",
              aes(x = longitude, y = latitude, shape = alt.type, fill=altitude, colour
              size=8, alpha=.8, width = .02, height = .02)+
  xlab("Longitude") + ylab("Latitude") +
  scale_shape_manual(values = c(24,22,21))+
  scale_fill_gradient( low = "green", high = "blue")+
  labs(size = "Elevation (m)") +
  theme(strip.background = element_blank(),
        panel.grid = element_blank()) +
  guides(size = guide_legend(order=2)); p1
```



Geographic distance

You can estimate simple geographic distance with the package ecodist (you will get a pairwise population geographic distance matrix)

```
library(ecodist)
era.ec.e.loc <- era.ec.e[,c("pop.name", "latitude", "longitude")]; era.ec.e.loc
era.ec.e.loc <- dplyr::summarise(dplyr::group_by(era.ec.e.loc, pop.name),latitude=mear
era.ec.e.dist <- earth.dist(as.data.frame(era.ec.e.loc[c("longitude", "latitude")]), c
```

The geographic distance between these populations is **78.8km**, but it is clear for the map that these butterflies have to overcome many obstacles to get from A to B.

Topographic distance

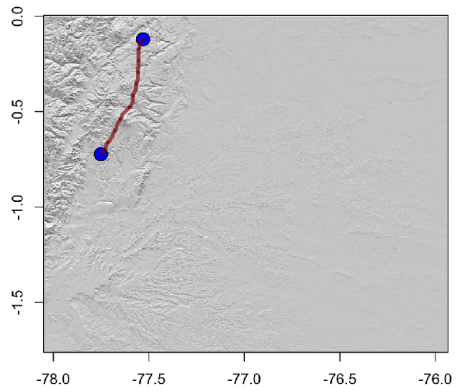
We will take advantage of a fantastic package called “topoDistance” (Wang, 2019, link) and elevatr (https://github.com/jhollist/elevatr).

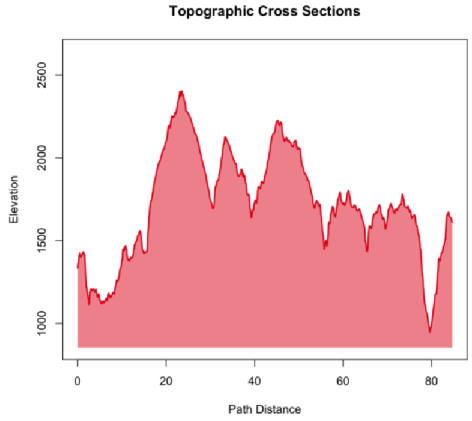
If we estimated the shortest “topographic distance” (topoDistance::topoDist()), it would draw lines between A to B, and add up the ‘ups and downs’, i.e. the topography- to obtain topographic distances. Or in better terms (from the topoDistance vignette):

> *The topoDist() function generates a topographic distance surface by calculating the topographic distance between cells as the hypotenuse of their horizontal and vertical distances. These distances are assignd to the weights of vertices between the nodes for each cell on a landscape graph. topoDist() then calls on functions in the gdistance (van Etten, 2015) and igraph (Csardi & Nepusz, 2016) packages.*

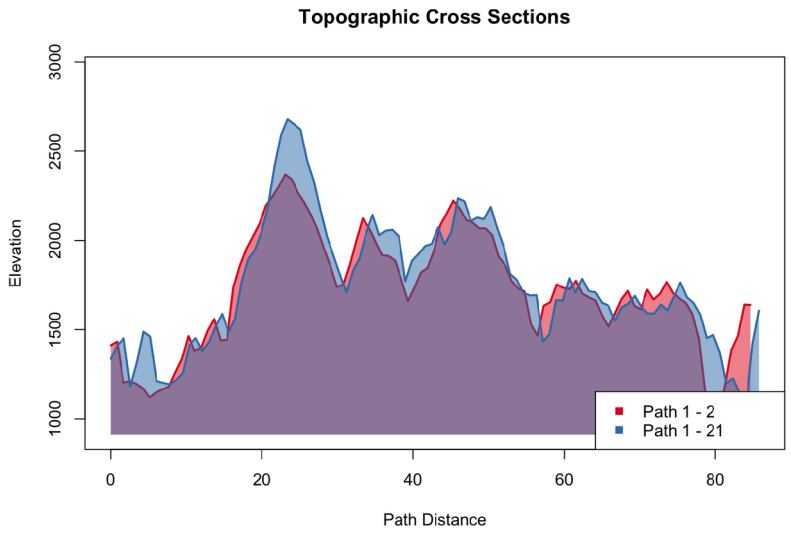
Then it would chooses the ‘paths’ that add up to the shortest topographic distance. These can be nicely visualised. Furthermore, you can do this for a set of populations/locations and obtain a ‘topographic distance matrix’.

```r
library(topoDistance); library(elevatr); library(raster)
# use era.ec.e.dem obtained with elavatr for the map in the preamble
# get topographic distance, paths=TRUE to then plot paths
tdist <- topoDist(era.ec.e.dem$layer, era.ec.e.df[,2:3][3:4,], paths = TRUE); tdist

# LEFT: plot map with shortest topographic paths and the landscape
tdist <- topoPathMap(era.ec.e.dem$layer, era.ec.e.df[,2:3][3:4,], topoPaths = tdist, t
            pathWidth = 4, cex = 2, bg = "blue"); tdist

# RIGHT: topograpic cross section of shortest topographic distance between A and B
topoProfile(era.ec.e.dem$layer, topoPaths = tdist, pts = 1000,
            type = "base", singlePlot = TRUE)

# BOTTOM: compare paths of shortest topographic distance with penalised shortest topog
# doesnt really
 twd <- topoWeightedDist(era.ec.e.dem$layer, era.ec.e.df[,2:3][3:4,], hFunction = "lir
                         vFunction = "exponential", paths = TRUE) #penalises changing e
topopaths <- rbind(tdist[[2]][1], twd[[2]][1])
topoProfile(era.ec.e.dem$layer, topopaths, type = "base",singlePlot = TRUE)
```





(comparing paths crosssections can be useful, here it is clear that the penalisation on movement along an incline is not sufficient to prevent paths going above the elevational range of the butterfly.)



The shortest topographic distance between A and B is **85km**, but from the cross sections we can detect that these paths repeatedly go above the elevational range of Heliconius erato (0-1600m.a.s.l)**.** So we need a better way of predicting the path a butterfly might take to go from A to B in the real world.

Topographic Least Cost Paths

For this we want to estimate the "landscape's resistance to movement", in other words, the aspects of the landscape that might prevent our focal species from moving from one cell to the next. In essence, how suitable the landscape is. When studying species distrubutions an easy way to look at suitability landscape, rather than trying to predict all the variables that make a landscape more habitable, is to create species distrubution models- SDM (i.e. what landscapes the butterfly actually inhabits!). Once we have a SDM, we can input this raster (map) to topoDistance::topoLCP() as a conductance surface (the inverse of resistance surface) to eastimate ' the easiest path for this butterfly species from A to B, given their natural distribution'. As you might predict this will not include flying over a volcano...

Briefly the pipeline would consist of:

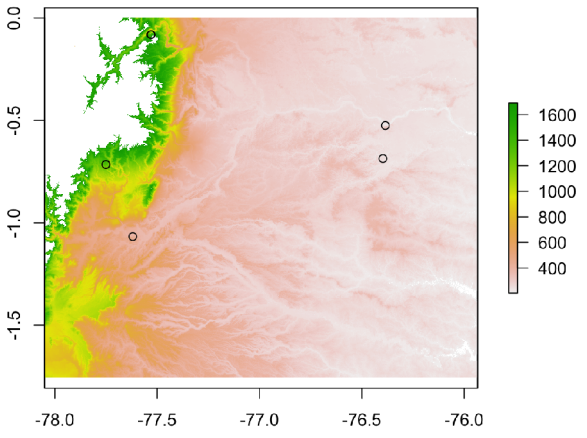
1. Obtain records of study species (localities), here Heliconius erato,
2. Create Species Distribution Model, visualise and save as a raster layer
3. Input SDM as costSurface to topoLCP()

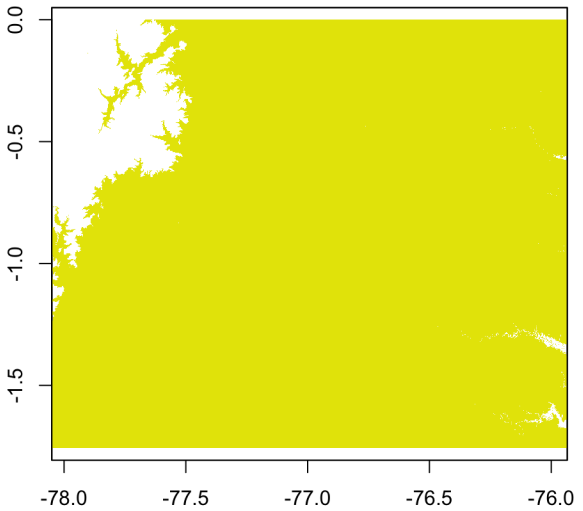
However step 2 is incredibly complicated and a whole field in itself, certainly beyond the scope of what we want to do here (sampling bias, model fits...this is a nice vignette on the sort of pipeline needed- link1 link2)

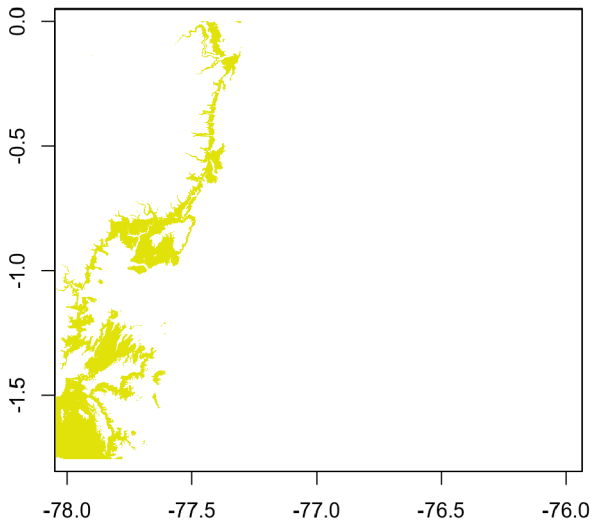
Im going to make step 2 much simpler, given historic sampling records (and my own) we are going to **create a binary habitat suitability raster** of Heliconius erato based only on one (but very important) factor- **elevational range**.

I have adapted this function so that from any elevational raster (edm created for example with elevatr), we can output a binary raster (if binary=TRUE) within an elevational range (with upper and lower thresholds), that can be either inputted manually (with threshold.upper=, threshold.lower=, type="manual") or taken from a set of points (species occurences: points=, type="max.min"). You can also add a buffer, which would expand the threshold (either manual or max.min) by the buffer%.

```r
raster_threshold <- function(input_raster, points = NULL, threshold.lower = NULL, thre
  raster_thresh <- input_raster
  if (type=="max.min") {
    pointVals <- raster::extract(input_raster, points)
      threshold.l <- min(na.omit(pointVals))- (min(na.omit(pointVals))*buffer/100)
      threshold.u <- max(na.omit(pointVals))+ (max(na.omit(pointVals))*buffer/100)
    } else if (type=="manual"){
        threshold.l <- threshold.lower- (threshold.lower*buffer/100)
        threshold.u <- threshold.upper+ (threshold.upper*buffer/100)
   }
  raster_thresh[raster_thresh < threshold.l] <- NA
  raster_thresh[raster_thresh > threshold.u] <- NA
  if (binary==TRUE) {
    raster_thresh[raster_thresh > threshold.l] <- 1
    raster_thresh[raster_thresh < threshold.u] <- 1
  }
  return(raster_thresh)
}

# LEFT: to check how it works, with points, binary=F - will show altitudes
edm.thresh.raster <- raster_threshold(input_raster = era.ec.e.dem, points = era.ec.e.d
points(era.ec.e.df[,2:3]) # add occurences used to create the thresholds

# RIGHT: with points, binary=T
bin.thresh.raster <- raster_threshold(input_raster = era.ec.e.dem, points = era.ec.e.d

# BOTTOM: manual thresholds, if we were interested in a certain range
man.thresh.raster <- raster_threshold(input_raster = era.ec.e.dem, type = "manual", th

# optional, obtain historical records from any species on the gbif database (as it hap
# ext is a raster box (area of map) to focus on, use extent from elevation map
erato = gbif("heliconius", "erato*", geo=TRUE, ext = era.ec.e.dem@extent); names(erato
```

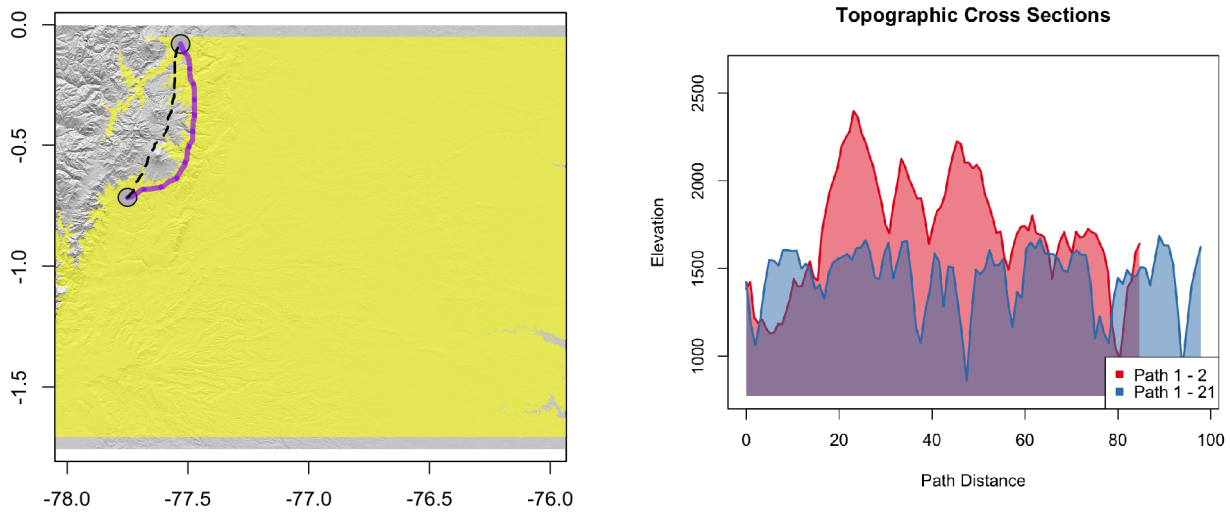






So now we can use this rough “species distribution” (**binary habitat suitability raster**, to be precise) to topoDistLTC

```
# use a binary threshold, base it on our own collections/elevation records for this sp
bin.thresh.raster <- raster_threshold(input_raster = era.ec.e.dem, points = era.ec.e.d

# find the least cost path
tLCP <- topoLCP(era.ec.e.dem$layer, costSurface = bin.thresh.raster, pts = era.ec.e.df
# LEFT: plot it in purple, with the costSurface (elevational range) as a background la
topoPathMap(era.ec.e.dem$layer,  era.ec.e.df[,2:3][3:4,], topoPaths = tLCP, type = "hi
            costSurface = bin.thresh.raster, pathWidth = 4, pathColor = "purple")
# plot the shortest topographic distance path to see difference, black
lines(tdist[[2]], lty = 2, lwd = 2

# RIGHT: compare cross section of topo dist and TLCP
topopaths <- rbind(tdist[[2]][1], tLCP[[2]][1])
topoProfile(era.ec.e.dem$layer, topopaths, type = "base", singlePlot = TRUE)
```



So the topographic least cost path from A to B (purple) is **99.7km** (compared to geographic distance of 78km and topographic distance -black dotted line- of 85km). On the cross sections on the right we can see that the least cost path (blue) is longer, hence the longer x-axis, but does not require going above the elevational range limit of this species.

Here I have only inputted 2 points (A and B), but it can be as many as we like, it will return a topographic least cost path matrix. Careful: very memory intensive, see topoDist paper for estimations, I'm running 12 populations on a HPC cluster. Memory depends on your input raster, in this case era.ec.e.dem- so play around with "zoom" and "clip" options from get_elev_raster to not get tiles hat are too small (too much resolution, lots of memory) or too big (too little resolution, jiggered paths).