Oikos

OIK-02426

Arellano, G., Tello, J. S., Jørgensen, P. M., Fuentes, A. F., Torrez, M. I. L. V. and Macía, M. J. 2015. Disentangling environmental and spatial processes of community assembly in tropical forests from local to regional scales. – Oikos doi: 10.1111/oik.02426

Appendix 1

Methods A1

Standardization of cation measurements between the ammonium-acetate and the Mehlich-3 extraction methods

We measured exchangeable cations using two extractive methods for different sets of samples from the plot network: 1) the 1 M ammonium acetate solution method and 2) the Melhich-3 extraction method (Mehlich 1984). These two methods are often strongly and linearly correlated in many types of soils (Eckert and Watson 1996), and we used this fact to stantardize all our measurements into comparable values. To do so, we analized with both methods a set of 76 soil samples chosen to span the entire range of variation in cation concentrations in the soils of the study region.

We then fitted linear ordinary least-square models to the relationships between values in the two methods, producing the following regression equations (Fig. A1):

Calcium: $Ca_{Melhich\text{-}3} = 53.032 + 0.3588 \times Ca_{ammonium\ acetate}$; $R^2 = 0.87$, $p < 0.001$

Magnesium: $Mg_{Melhich\text{-}3} = -49.387 + 0.6635 \times Mg_{ammonium\ acetate}$; $R^2 = 0.78$, $p < 0.001$

Potassium: $K_{Melhich\text{-}3} = 28.689 + 0.7326 \times K_{ammonium\ acetate}$; $R^2 = 0.68$, $p < 0.001$

With these equations, we transformed all ammonium acetate values into their Mehlich-3 equivalents. Sodium was not included in the analysis because it produced a weak and non-significant relationship between methods (Fig. A1).

References

Eckert, D. J. and Watson, M. E. 1996. Integrating the Mehlich -3 extractant into existing soil test interpretation schemes. – Comm. Soil Sci. Plant Anal. 27: 1237–1249.

Mehlich, A. 1984. Mehlich 3 soil test extractant: a modification of Mehlich 2 extractant. – Comm. Soil Sci. Plant Anal. 15: 1409–1416.

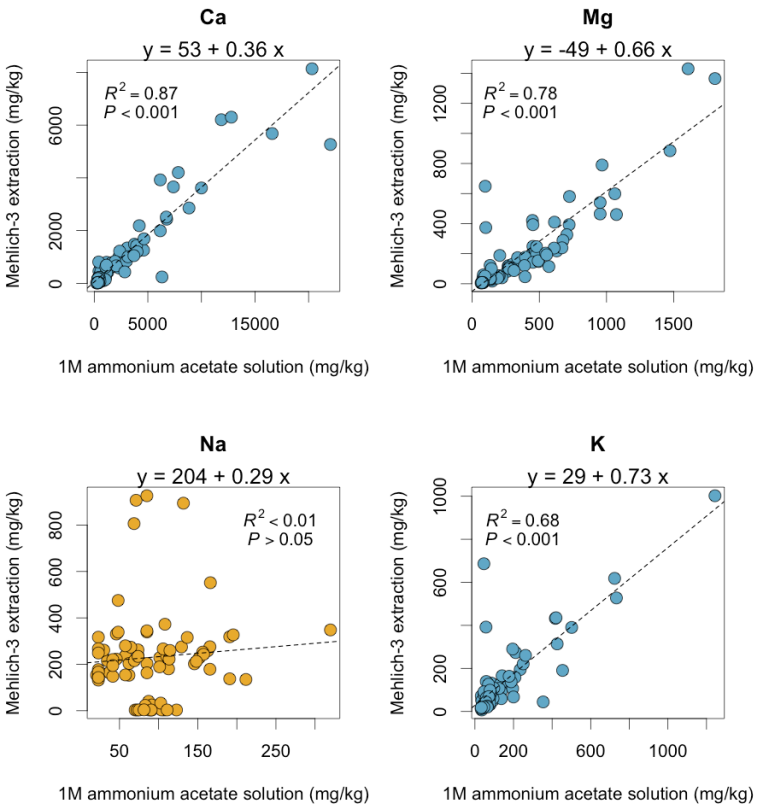

Figure A1. Relationships in cation concentrations between the 1 M ammonium acetate solution and the Melhich-3 extraction methods.

Methods A2

Construction of sub-regions with varying spatial extent

To investigate the effects of spatial extent (i.e. size of regions, spatial scale), we constructed 200 sub-regions based on portions of the dataset within our study region. Each sub-region was defined as a unique set of 60 plots, while spatial extent was measured as the area of the minimum convex polygon that contained all the 60 plots. The selection of plots and of sub-regions for our analyses was not completely random. Instead, we used an algorithm to guarantee the equitable representation of regions of different extents. In this way, analyses would not be dominated by small sub-regions, which are the easiest to construct (Fig. A2). Furthermore, our algorithm also minimized the overlap between sub-regions in term of the plots they contained, so that no sub-region shared more than 70% of its plots with any other sub-region. The final average proportion of shared plots was less than 10% (Fig. A2). Finally, our algorithm also produced a set of sub-regions where spatial extent and average elevation were not correlated (Fig. A2). This guarantees that elevation is not a confounding factor in our analyses of spatial extent.

The algorithm we used can be described in the following steps:

1. Define seven[1] squared areas centered at each of the 398 plot in the dataset (i.e. 2786 initial areas). These areas range in size from 1 × 1 to 196 × 196 km. The size of the whole study region was 158 × 196 km.
2. Eliminate any areas that do not contain at least 60 plots.
3. Create a first set of candidate sub-regions by choosing 60 plots at random within each squared area.
4. Calculate the spatial extent of candidate sub-regions as the area of the minimum convex polygon containing all 60 randomly selected plots.
5. Sort candidate sub-regions by extent from small to large.
6. Create a second set of candidate sub-regions by eliminating sub-regions with a large number of shared plots. For this step, each candidate sub-region is analyzed individually in increasing order of spatial extent. A sub-region is kept if it shares less than 70% of its plots with other sub-regions already selected. If it shares 70% or more, the sub-region is eliminated. If during this process, the algorithm has already gone over the first 20% of the candidate sub-regions *and* has discarded less than 25% of the sub-regions evaluated, then all remaining sub-regions are kept and become part of the second set of candidate sub-regions. This was done to reduce computational time for the evaluation of large sub-regions. Large sub-regions are already unlikely to share many plots with other sub-regions, because of the extremely large number of

[1] Determined by applying the Sturge's rule: $k = \lceil log_2 n + 1 \rceil$; thus for $n = 60, k = 7$

random samples of 60 plots that can be obtained from a large pool of plots in Step 3.

7. Group the second set of candidate sub-regions by spatial extent into 15 equal-range classes.
8. Create a final set of sub-regions. For this, the algorithm iterates over each class of spatial extent extracting at random one sub-region per class. This process is repeated without replacement until 200 sub-regions have been selected.

The algorithm was implemented using two functions written in R: `calculateScale` (a function that calculates the scale of a given set of points, according to a given definition of "scale") and `selectSubregions` (the main function).

The `calculateScale` function takes the following arguments:

- `subregions`: a boolean matrix with one row per sample and one column per sub-region, indicating whether a plot belong to a sub-region or not.
- `coor`: a matrix with the coordinates of the plots in the sub-regions, in decimal degrees.
- `e`: the extent criterion to be employed: "`mcp`" for the area of the minimum convex polygon, "`nndist`" for the mean nearest neighbor distance, "`spantree`" for the mean distance between neighbors in a minimum spanning tree, "`max`" for the maximum distance plots, "`mean`" for the mean distance between plots.

The function returns a numeric vector with the extent of the sub-region in km or km^2 (depending on the extent criterion specified).

```
calculateScale <- function(subregions, coor, e = c("mcp",
"nndist",
  "spantree", "max", "mean")) {

  require(spatstat)
  require(adehabitatHR)
  if(length(e)>1) e <- "mcp"

  scale <- numeric(ncol(subregions)) # empty vector
  if(e == "mcp")
    for(i in 1:ncol(subregions))
      scale[i] <-
        (mcp(SpatialPoints(coor[subregions[,i],]),
```

```
unin="km",
          unout="km2", percent=100)$area)

  if(e == "nndist")
    for(i in 1:ncol(subregions))
      scale[i] <- mean(nndist(coor[subregions[,i],]))

  if(e == "spantree")
    for(i in 1:ncol(subregions))
      scale[i] <-
mean(spantree(dist(coor[subregions[,i],]))$dist)

  if(e == "max")
    for(i in 1:ncol(subregions))
      scale[i] <- max(dist(coor[subregions[,i],]))

  if(e == "mean")
    for(i in 1:ncol(subregions))
      scale[i] <- mean(dist(coor[subregions[,i],]))

  scale <- scale*111.111 # from degrees to km
  if(e == "mcp")
    scale <- scale*111.111 # …or km²

  return(scale)
}
```

The `selectSubregion` function takes the following arguments:

- `coor`: matrix with the coordinates of the samples in decimal degrees, with "`long`" and "`lat`" columns.
- `extent.criterion`: same as `e` in the `calculateScale` function (see above).
- `N`: number of desired sub-regions.
- `minimum.n`: minimum number of samples within each sub-region.
- `maximum.n`: maximum number of samples within each sub-region. If not specified, it takes the same value as `minimum.n`.

- `sim.threshold`: a threshold above which sub-regions are considered "too similar" to be included. For example, if `sim.threshold=0.70`, The function will drop sub-regions until no pair of sub-regions share more than 70% of its plots.
- `breaks`: break points for the scale classes (like in a histogram).

And it returns a list with the following components:

- `bool`: a boolean matrix indicating if a plot is included or not within a sub-region.
- `scale`: a vector with the extent, following the indicated criterion for extent.
- `extent.criterion`: the criterion employed.
- `scale.units`: the units of scale.
- `sim.threshold`: the parameter employed.
- `h`: an histogram object, useful to extract parameters with which visualize how the algorithm has selected different scales.

```
selectSubregions <- function(coor,
extent.criterion=c("mcp", "nndist",
  "spantree", "max", "mean"), N=100, minimum.n=50,
  maximum.n=minimum.n, sim.threshold=0.7, breaks=15) {
   require(vegan)

  # Starts using the Sturges' rule to perform
  # a preliminar search of sub-regions of different sizes:
  z <- max(max(coor[,1])-min(coor[,1]), max(coor[,2])-
min(coor[,2]))
 length.sides <- as.list(seq(from = z/200, to = z,
    length.out = ceiling(1 + log(nrow(coor), base=2))))

   # A first search:
  L <- list()
  for(i in 1:nrow(coor)) { # for each sample...

    centre.x <- coor[i,1]; centre.y <- coor[i,2]

    corners.coordinates <- lapply(length.sides,
function(x)
```

```
      c(xmin=centre.x-x/2, xmax=centre.x+x/2,
ymin=centre.y-x/2,
      ymax=centre.y+x/2))

    which.within <- lapply(corners.coordinates,
function(x)
      which(coor[, "long"]>=x["xmin"] & coor[,
"long"]<=x["xmax"] &
      coor[, "lat"]>=x["ymin"] & coor[,
"lat"]<=x["ymax"]))

    which.suitable <-
      which.within[lapply(which.within, length) >=
minimum.n]

    L <- c(L, lapply(which.suitable, function(x)
      sort(sample(x, size=sample(rep(c(minimum.n :
min(c(maximum.n,
      length(x)))), 2))[1])))) # randomness enters here
  }

  # Goes from a list to a boolean matrix, to eliminate
quickly
  # repeated combinations of plots (from indexes to
boolean).
  subregions <- matrix(FALSE, nrow=nrow(coor),
ncol=length(L),
    dimnames=list(rownames(coor), NULL))

  for(i in 1:length(L))
    subregions[L[[i]],i] <- TRUE

  ########################################################
  ### The following debugs the sub-regions matrix      ###
  ### to make it meet the specified conditions         ###
  ########################################################
```

```
  subregions <- unique(subregions, MARGIN=2) # removes
duplicates
  scale <- calculateScale(subregions, coor=coor,
e=extent.criterion)

  # 1. Removes "similar" sub-regions (above the specified
  # Jaccard similarity threshold). This can be super-slow,
  # so there is the option of making forward inclusion
  # from small to large scales.
  subregions <- subregions[,order(scale)] # sorted by
scale
  subregions <- apply(subregions, 2, which) # boolean to
indexes

  if(!is.list(subregions)) { # just to ensure this is a
list
    colnames(subregions) <- 1:ncol(subregions)
    subregions <- split(t(subregions),
colnames(subregions))
    names(subregions) <- NULL
  }

  L <- list(subregions[[1]])
  count <- 1
  for(i in 1:length(subregions)) {
    candidate <- subregions[[i]]
    if(sum(unlist(lapply(L, function(x)
      length(intersect(candidate, x)) /
      length(union(candidate,x))>=sim.threshold)))==0) {
        count = count+1
        L[count] <- list(subregions[[i]])
    }

    # By activating the following line of code,
    # it saves time by increasing a little bit
    # the probability to include not dissimilar
```

```
    # enough sub-regions:
    if(i>0.2*length(subregions) & count/i>0.75) break
  }

  # The following completes the rest with the remaining
"blindly"
  # It applies only if we choose to break the loop,
  # and may include a few "similar" sub-regions
  for(i in (count+1):length(subregions))
    L[i] <- list(subregions[[i]])

  # Again returns to boolean and calculates scale:
  subregions <- matrix(FALSE, nrow=nrow(coor),
ncol=length(L),
    dimnames=list(rownames(coor), NULL))

  for(i in 1:length(L))
    subregions[L[[i]],i] <- TRUE

  scale <- calculateScale(subregions, coor=coor,
e=extent.criterion)

  # 2. Selection to represent the different scales
  # as equitably as possible:
  h0 <- hist(scale, breaks=breaks, plot=FALSE)
  counts <- h0$counts # observed counts
  breaks <- h0$breaks # break points for the scale classes

  # Defines the desired count for each scale class
  desired.count <- numeric(length(counts)); i = 0
  while(sum(desired.count) < N) {
    i = i+1
    desired.count <- rep(i, length(counts))
    desired.count[which(counts<i)] <-
counts[which(counts<i)]
  }
```

```
  # Picks samples according to the desired count
  chosen <- NULL
  for(i in 1:length(desired.count)) {
    lower = breaks[i]
    upper = breaks[i+1]
    if(i==1)
      good.ones <- which(scale>=lower & scale<upper)
    if(i!=1)
      good.ones <- which(scale>lower & scale<=upper)

    a.few.good.ones <- sample(good.ones,
size=desired.count[i])
    chosen <- c(chosen, a.few.good.ones)
   }
   chosen <- unique(chosen)

  # 3. Creates output:
  bool <- subregions[,sample(chosen, min(length(chosen),
N))]
  scale <- calculateScale(bool, coor=coor, e =
extent.criterion)

  if(length(extent.criterion)> 1)
    extent.criterion <- "mcp"
  if(extent.criterion == "mcp")
    scale.units = "km2"
  if(extent.criterion == "nndist")
    scale.units = "km"
  if(extent.criterion == "spantree")
    scale.units = "km"
  if(extent.criterion == "max")
    scale.units = "km"
  if(extent.criterion == "mean")
    scale.units = "km"
```

```
  out = list(bool = bool, scale = scale,
    extent.criterion = extent.criterion, scale.units =
scale.units,
    sim.threshold = sim.threshold, h=h0)

  return(out)
}
```

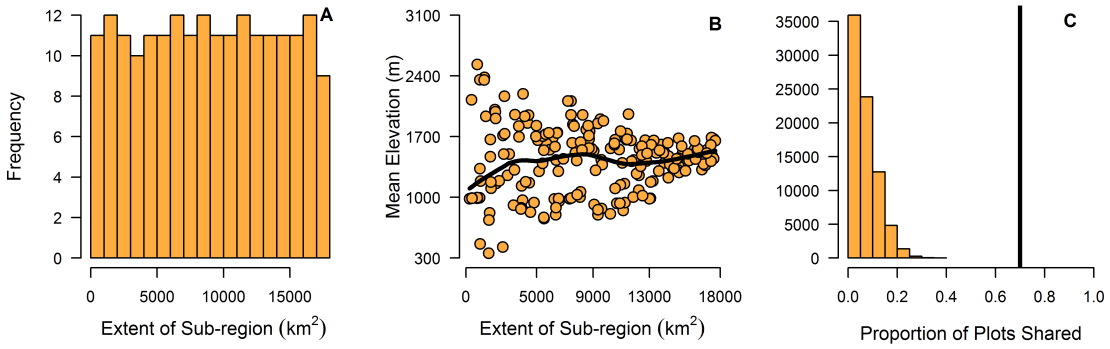

Figure A2**.** Results of the algorithm to construct 200 sub-regions with varying spatial extent. (A) Flat histogram of spatial extents (all spatial scales represented similarly). (B) No relationship between elevation and spatial extent. Line shows the fit of a locally-weighted polynomial regression. (C) Low overlap (proportion of shared plots) between pairs of sub-regions. The vertical line indicates the similarity threshold in the algorithm of 70% overlap.

Methods A3

Heterogeneity and richness within sub-regions

We were also interested in understanding how environmental heterogeneity and total richness changed across sub-regions as a function of spatial extent. To calculate environmental heterogeneity, we first standardized the values of all environmental predictors across all plots to a mean of zero and a standard deviation of one. Then, for the plots in each sub-region, we calculated the mean Euclidean distance in the multidimensional space defined by all climatic or soil predictors. We used this mean distance as our measure of environmental heterogeneity. To calculate elevational heterogeneity, we simply used the range in elevation among the plots within a sub-region. Finally, we counted the total number of species across the 60 plots in each sub-region. As expected, we found that climatic, soil and elevational heterogeneity increased with increasing spatial extent (Fig. A3). However, there was a good amount of scatter in these relationships, and even small regions could have large values of heterogeneity. Richness had a much tighter relationship with spatial extent (Fig. A3), but even small regions had large numbers of species (~400). The regions with the largest spatial extents could have more than 1250 species.

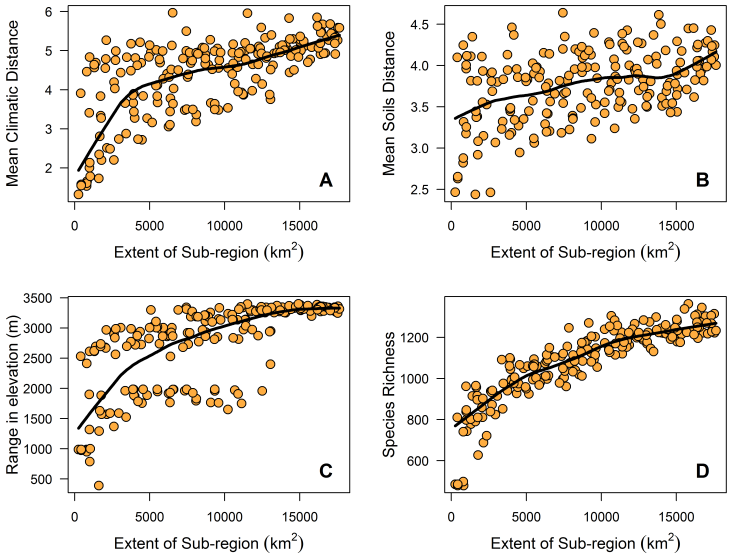

Figure A3. Environmental heterogeneity and richness of sub-regions as a function of spatial extent: (A) climatic heterogeneity, (B) soils heterogeneity, (C) range in elevation and (D) species richness. Black lines shows locally-weighted polynomial regressions fitted to each relationship.