Online Appendix for "Representing Ethnic Groups in Space: A New Dataset"

Journal of Peace Research, 2010, in press.

NILS B. WEIDMANN
International Conflict Research, ETH Zurich

JAN KETIL RØD
Department of Geography, Norwegian University of Science and Technology
& Centre for the Study of Civil War, PRIO

LARS-ERIK CEDERMAN
International Conflict Research, ETH Zurich

General Description of the Dataset Format

The GREG dataset is provided in the shapefile format (ESRI, 1998). A shapefile is one of the most common formats for GIS vector data and can be read and processed by almost any GIS package, as well as some statistical packages with extensions for spatial data (see our example below). Basically, a shapefile contains a data table with a set of variables, similar to all data formats used in statistics. This table is called the *attribute table*. The lines in the attribute table are connected to individual geographic features, i.e. the spatial entities represented in the dataset such as cities, rivers or lakes, such that each row in the attribute table corresponds to exactly one feature (or multipart feature). These features are stored with their precise coordinates in a given projection and coordinate system. Thus, the shapefile data format allows for the representation of both the geographic entities of interest, and the non-spatial information associated with them. For example, a shapefile could consist of a set of points representing cities, with the non-spatial information about these cities (city name, population etc.) stored in the corresponding records in the attribute table. The features in the GREG dataset are polygons representing the settlement regions of groups, according to the ANM. The records in the attribute table store the information corresponding to these polygons, such as the identifier of the polygon and the polygon area, but most importantly, the names and identifiers of the groups living in the

respective settlement region. A detailed description of the attributes is provided below.

Geographic Reference System

The GREG shapefile uses geographical coordinates (latitude/longitude) based on the World Geodetic System (1984 specification, WGS84). Its geographic extent ranges from longitude -179.99W to 179.99E, and latitude -55.31S to 82.21N.

Attributes

The GREG attribute table stores the non-spatial information associated with the polygons. The attributes are as follows.

G1ID, G2ID, G3ID (Type: Short integer): The unique group IDs of the groups in the polygon, and 0 if not applicable. Every polygon in GREG can be occupied by up to three groups. Since every polygon represents at least one group, G1ID never takes the value 0. Polygons with a single group have G2ID=0 and G3ID=0, and polygons with two groups have G3ID=0.

GxSHORTNAM, GxLONGNAM (Type: Text): Three columns each for the short names and the long names of the group(s) in the polygon, using the translated group names in the ANM. For the user's convenience, we also provide a separate text file with a complete list of the groups and their IDs.

FeatureID (Type: Integer): A unique identifier for the polygon.

AREA (Type: Double): The polygon area in square meters, computed on an Eckert VI equal area projection with metric coordinates.

COW (Type: Integer): The COW code of the country the given polygon belongs to, based on the 1989 international system.

FIPS_CNTRY (Type: Text): The *Federal Information Processing Standard* (FIPS) is an alternative coding scheme for geographic units, used in many GIS applications. This attribute contains the two-character FIPS code of the country a polygon belongs to.

Details about the ANM Maps and the Geo-referencing procedure

The reference system used for the ANM maps are geographical coordinates (latitudes and longitudes) with Greenwich as the Prime Meridian (longitude = 0°). The map datum and the projection are not given. We scanned the maps from the *Atlas* and saved the individual maps as JPEG images. These images were imported into ArcGIS, where we geo-referenced them based on a number of ground control points (GCP). A ground control point is a point with known coordinates both in the image coordinate system and in the target coordinate system, which in our case are geographical coordinates with the WGS84 datum. The GCPs were distributed uniformly in the scanned maps (with points along the edges as well as an even distribution within). Since a graticule is present on all the ANM maps, it was convenient to use latitude/longitude intersections as GCPs. In addition, clearly identifiable points such as border junctions and intersections were used as GCPs. For the georeferencing of the scanned maps we used an affine transformation to allow for uneven shrinkage along the first and second reference axis. Given that a sufficient number of GCPs are registered, ArcGIS allows the use of a transformation polynomial of first, second and third order. The required number of GCPs for a first, second and third order polynomial is 3, 6 or 10. We registered more than the minimum required, usually around 20 GCPs per map. For all the maps we did a visual control of the fit between the scanned map with other reference features (such as international boundaries), as well as an assessment of the fit of the transformation (typically requiring the RMS error to be below 1.0 pixels). As ArcGIS also reports the residual for each of the entered GCPs (thus its contribution to the overall RMS error), we removed GCPs with high residuals and replaced them with new ones. When we obtained a good fit, we started the screen digitizing.

The scale of the ANM maps ranges from 1:4 000 000 up to 1:15 000 000. What is the level of accuracy we can expect from these maps? Although we have done the georeferencing and screen digitizing as accurate as we could, a useful rule of thumb says that positions measured on maps are accurate to about 0.5 mm on the map (Longley et al., 2005: 142-143). Multiplying this by the scale of the map gives the corresponding distance on the ground. For a map at scale 1: 15 000 000, the ground distance corresponds to 7.5 kilometers. For a conservative assessment we could

therefore say that the information in GREG can deviate from the ANM within a range of about 10 kilometers.

How to Use the GREG Data

In the following, we give two short examples how to use the GREG data in practical applications. Some of the more advanced computations with GREG require good GIS skills and would therefore be beyond the limits of this paper. However, this is not the case for many of the simpler operations involving GREG data. This short introduction relies on R, a freely available statistical package.[1] R comes with quite a number of free extension packages for spatial computing, which also allow for the processing of the shapefile format used for GREG. For the installation of R and the required extension packages we refer the reader to the general R installation manual, available on the R web page. The detailed replication code for the given examples is available on the web page that accompanies this article. The following paragraphs do not contain all the required R commands, as we focus on the most important ones.

The extension packages provide additional functionality to R such that a shapefile can be used almost in the same way as an R data frame, which is the basic R data type used for storing data matrices in statistical analysis. Having read the shapefile's attribute table into such an R spatial data frame, basic operations such as selecting records and columns or computing summary statistics can be executed in the same way as for a normal, non-spatial data frame. Attached to the individual records in the spatial data frame are the corresponding geographic features, i.e. the polygons in the case of GREG. Our first example shows how to select a subset of features from a spatial data frame, and use then for drawing an ethnic map.

Example 1

We use the readShapePoly command to load a shapefile into R. The shapefile (with all its additional files, as downloaded from the GREG website) must be located in the R working directory, or the full path to the dataset must be given. We specify the map datum and the coordinate system of the GREG dataset in the command.

[1] R is available at http://www.r-project.org/. The examples we present were developed using R version 2.7.1 and require the `sp` and `spdep` extension packages for R.

```
greg <- readShapePoly("GREG.shp",
proj4string=CRS("+proj=longlat +datum=WGS84"))
```

Using the summary command, we can output the summary statistics for all variables in GREG.

```
summary(greg)
```

Now, we start the creation of the ethnic map for Switzerland. First, we select all group polygons for Switzerland, filtering on COW code 225. This operation is exactly the same as for every R data frame.

```
switzerland <- greg[greg$COW==225,]
```

We can also use R's built-in functions, as for example for computing the mean area (in square kilometers) for the group polygons in Switzerland.

```
mean(switzerland$AREA)/1000000
```

To get nice labels with the group names for our map, we concatenate the group names contained in the three GxSHORTNAM fields in GREG into a single string. Since every polygon contains at least one group, G1SHORTNAM is never empty. However, for all polygons with one or two groups, at least one of the columns G2SHORTNAM and G3SHORTNAM is empty (coded as "NA" in R), so we need to check for this to get a nicely formatted string of group names.

```
g2name <- ifelse(switzerland$G2ID==0, "",
     paste(", ",switzerland$G2SHORTNAM, sep=""))
g3name <- ifelse(switzerland$G3ID==0, "",
     paste(", ",switzerland$G3SHORTNAM, sep=""))
groupnames <- paste(switzerland$G1SHORTNAM,
     g2name, g3name, sep="")
```

Now, we are ready to plot the polygons. Plotting a shapefile in R is simple, we can just use the normal plot command.

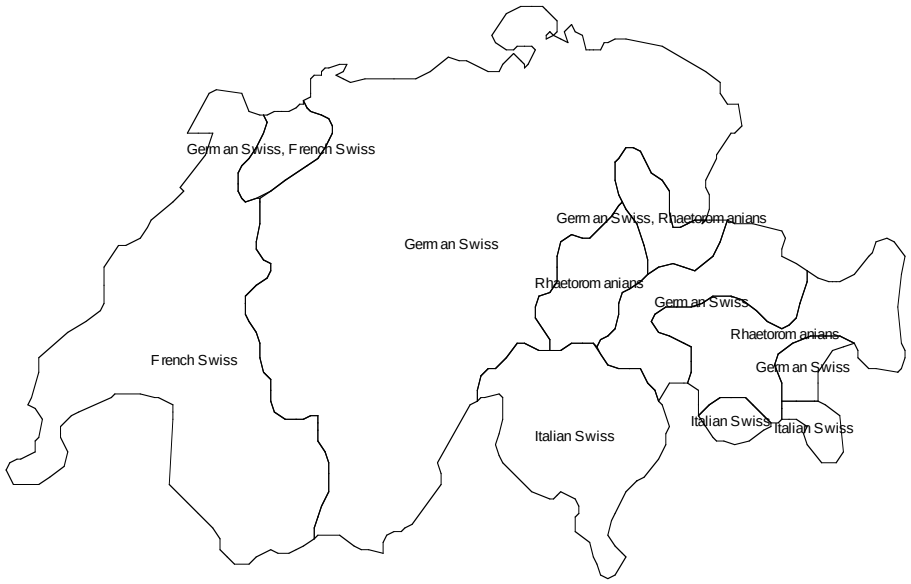
```
plot(switzerland)
```

We finish the plot by adding the group names created above. Notice the use coordinates functions that when applied to a polygon dataset, returns the centroids of the polygons. We use these centroids to position the group labels.

```
text(coordinates(switzerland),
     labels=groupnames, cex=0.6)
```

The result is an ethnic map of Switzerland, shown in Figure A1. More fine-tuning of the map is also possible, as for example color shading by group regions.

Figure A1: Ethnic Map of Switzerland, Created in Our First Example



Example 2

Our second example demonstrates the computation of a simple indicator for group concentration based on GREG. The underlying idea is as follows. A group is concentrated if it settles in a single contiguous region of a country. If the group occupies only one polygon in a country, it is considered as concentrated. If it occupies more than one polygon, two possibilities exist. First, all these polygons can be located close to each other such that they form a single contiguous cluster. In this case, we would also code the group as concentrated, since even though the group occupies multiple polygons, there is only one greater settlement area of the group. Second, the group polygons can be located far away from each other, in which case we would code the group as dispersed.

Our concentration indicator retrieves all group polygons of a group in a country and groups them into clusters, where a cluster is a set of polygons that are contiguous to each other. The number of these clusters is used as an independent variable in a regression analysis. The following steps describe the computation of the number of clusters. Again, we focus only on the most important loop in the R program. The complete code can be found in the replication file provided for this article.

We begin with a for-loop that iterates through all countries in the dataset, identified by their COW code. We skip invalid COW codes (0).

```
for (cow in unique(greg$COW)) {
     if (cow==0) next
```

Next, we store all the group polygons that belong to the country we are currently looking at in a variable called `greg.cow`:

```
     greg.cow <- as.data.frame(greg[greg$COW == cow,])
```

We now need to find out which groups occur in the current country. To this end, we create three vectors, g1, g2 and g3 that contain the values of the GxID identifiers for the country polygons we have selected above (omitting "0" values). The union of these vectors is a list of group identifiers for the groups occurring in country `cow`.

```
     g1 <- unique(greg.cow$G1ID)
     g2 <- unique(greg.cow$G2ID[greg.cow$G2ID!=0])
     g3 <- unique(greg.cow$G3ID[greg.cow$G3ID!=0])
     groups <- union(union(g1,g2),g3)
```

Once we have obtained the list of groups for the current country, we compute the number of clusters for each of them.

```
     for (group in groups) {
```

First, we retrieve all the polygons where this group occurs in, either as a first, second or third group.

```
     greg.group <-
          greg.cow[greg.cow$G1ID==group
          | greg.cow$G2ID==group
          | greg.cow$G3ID==group,]
```

Second, we count the number of group polygons in the current country. If there is only one polygon, we can already now output the number of clusters for this group, 1.

```
     if(nrow(greg.group) == 1) {
          no.clusters <- 1
     }
```

If there are more polygons, however, we need to do group them into one or more clusters. We retrieve a new map of the group from the GREG dataset

```
  else {
       map.group <-
          greg[greg$COW==cow & (greg$G1ID==group
```

```
| greg$G2ID==group | greg$G3ID==group),]
```

On this map, we create a "neighbor list". This list simply tells us which of the polygons are contiguous to each other – all these polygons will then end up in the same cluster.

```
nblist <- poly2nb(map.group)
```

Next, we check the numbers of neighbors according to the neighbor list created in the previous step, using the card() function. If all these numbers are zero, this means that the group polygons are completely disconnected, and we can stop the computation here (see below). However, if at least one polygon has one or more neighbors, we need to group them into clusters. This is done by converting the neighbor list into an undirected graph, and running a provided cluster computation algorithm on it.

```
if (any(card(nblist) != 0)) {
```

We convert the neighbor list to an undirected graph…

```
     adjmat <-
      nb2mat(nblist,style="B",zero.policy=T)
     graph <-
      graph.adjacency(adjmat,mode="undirected")
```

… and compute the number of clusters in that graph:

```
     no.clusters <- no.clusters(graph)
}
```

However, if all group polygons are disconnected, we can simply output the number of polygons, since each of them constitutes a separate cluster.

```
else {
     no.clusters <- length(card(nblist))
}
```

Note that for the sake of illustration, we store the number of clusters in a single variable (`no.clusters`). See the replication code for how the values computed for this variable can be appended to a complete dataset as to use them for regression analysis.