**BIEN2 GeoValidation Workflow**

This document outlines the approach that was taken to validate the geographic (latitude and longitude) coordinates in the BIEN 2 database. The aim of the geovalidation process was the check the validity of the coordinates to ensure they were accurate and were representative of the location described in the locality description information that accompanies each species occurrence (whether an individual observation or a plot member).

The basic geovalidation approach was to intersect all species occurrences that contained geographic coordinates with shapefiles that contained country and state-province level geographies. Then we compared the attributes of the country and state-province obtained from the intersection with those recorded in the species’ locality description.

While this is conceptually simple, the process was complicated by several factors:

* The process required a spatial join that could not be reliably performed in MySQL due to the inaccuracies of the way that MySQL handles point in polygon operations. So we had to extract the species occurrences from MySQL and import them into a separate PostGIS database.
* As we had no PostGIS database at NECEAS at the time, we had to use our own hardware resources and used a PostGIS database on my laptop.
* The number of species occurrences needing processing outpaced our resources. So we had to break the geovalidation process into batches that required reassembly and reimporting back into MySQL afterwards.
* The spatial join in PostGIS could also not handle a large number of occurrences. So the spatial join was performed in ArcGIS and PostGIS was used only to calculate the geographic error (measured as the distance between a point and the country and/or state-province that it claimed to be from. A distance of 0 meant that the point’s coordinates were properly from the geographic unit they described, while any distance greater than 0 represented the distance (in kilometers) between the point and the nearest edge of the geography it claimed to be within.
* The basic idea of comparing geographic names (country name and/or state-province name) was complicated by the fact that the species occurrences often contained misspelled names, local variations of names, names recorded in the local language, or old names for geographic places that no longer existed. Therefore we had to standardize the country and state-province names and rely on comparing numeric IDs that were assigned to each geographic unit and recorded both in the geoscrub table of species occurrences and the shapefiles used for the spatial join.

**Outline of Processing Steps**

Obtain two spatial datasets (mine have edits that may be useful):

* GADM1\_Level0 (country boundaries)
* GADM1\_Level1 (state/province boundaries).

**In MySQL**

* Export a copy of the geoscrub, country and stateProvince tables from MySQL into PostGIS.

**In ArcGIS**

* Import GADM1\_Level0, GADM1\_Level1 spatial datasets into ArcGIS.
* Import country and state province datasets into ArcGIS.
* Add fields to GADM1\_Level0 spatial dataset to store the BIEN countryID (named “biencountryid” in my SQL code example below)
* Add fields to GADM1\_Level1 spatial dataset to store the BIEN stateProvinceID (named “bienstateid” in my SQL code example below).
* Join GADM1\_Level0 to country table by ISO code and populate GADM1\_Level0 table with our countryID
* Join GADM1\_Level1 to country table by HASC\_1 code and populate GADM1\_Level1 table with our stateProvinceID
* This associates countries and state-province polygons with our unique IDs for each jurisdiction.
* Import the geoscrub table into ArcGIS and convert to XY data for mapping.
* Save a copy of the geoscrub table as a shapefile or geodatabase feature class.
* Turn off the display of all fields in the GADM1\_Level0 and GADM1\_Level1 feature classes except those for the ISO, HASC\_1 codes and country and state-province names.
* Using the geoscrub shapefile or feature class perform an IDENTITY function on GADM1\_Level0 to copy the attributes of each country polygon to the geoscrub feature class.
* Repeat the IDENTITY function with GADM1\_Level1 to copy the attributes of each state-province polygon to the geoscrub feature class.
* Export GADM1\_Level0 and GADM1\_Level1 spatial datasets to shapefiles and import the shapefiles into your PostGIS database.

shp2pgsql -W latin1 -s 4326 -c -g the\_geom -I gadm1 gadm1\_level1 geoscrub > gadm1\_level1.sql

shp2pgsql -W latin1 -s 4326 -c -g the\_geom -I gadm1\_level0 gadm1\_level0 geoscrub > gadm1\_level0.sql

* Export the geoscrub table and import it into your PostGIS database.

**In PostGIS**

* Add a field to your copy of the geoscrub table to store a point (geometry) data type that will come from the latitude and longitude values in the geoscrub table.
* Populate the new point (geometry) field you just added using the following query:

update geoscrub

set

the\_geom=geomfromtext('POINT(' || "longitudeDecimalVerbatim" || ' ' || "latitudeDecimalVerbatim" || ')', 4326);

* Index the point field data using the following query:

CREATE INDEX geoscrub1\_the\_geom\_idx ON geoscrub1 USING gist(the\_geom);

* Perform batch spatial SQL queries for 10,000 species at a time (or some number that is manageable for your hardware and software configuration) to calculate the distance from a point to the polygon it reports to be within (distance is 0 if within and greater than 0 if outside).
* Example SQL code for validating country error:

SELECT "geoscrubID", "countryVerbatim", "countryID",

 "latitudeDecimalVerbatim", "longitudeDecimalVerbatim",

 ST\_Distance(o.the\_geom::geography,

 ST\_ClosestPoint(c.the\_geom, o.the\_geom))/1000 AS distance\_km

 FROM geoscrub1 AS o

 LEFT JOIN (

 SELECT "biencountryid", ST\_Collect(the\_geom) AS the\_geom

 FROM gadm1\_level0

 GROUP BY "biencountryid"

 ) AS c

 ON c."biencountryid" = o."countryID"

 WHERE

 (o."countryID" > 0) AND

 (o."longitudeDecimalVerbatim" IS NOT NULL) AND

 (o."latitudeDecimalVerbatim" IS NOT NULL) AND

 ((o."longitudeDecimalVerbatim" <> 0) AND (o."latitudeDecimalVerbatim" <> 0)) AND

 ((o."longitudeDecimalVerbatim" >= -180) AND (o."longitudeDecimalVerbatim" <= 180)) AND

 ((o."latitudeDecimalVerbatim" >= -90) AND (o."latitudeDecimalVerbatim" <= 90))

ORDER BY o."geoscrubID"

* Export the completed geoscrub table from PostGIS and re-import it back into MySQL.

**In MySQL**

* Continue with Brad’s geovalidation step 4 bullet 7.