Retrospective Georeferencing for Canadian Census Subdivisions in BC

Nathan G. Earley

2025-05-28

Intro

This document gives the rationale and background behind my retrospective georeferencing methods for insect specimens in the entomology collections in British Columbia (BC), Canada. The protocol that I have followed throughout this exercise is Zermoglio et al. (2020). This is the suggested protocol for uploading data to GBIF (Chapman & Wieczorek 2020).

Packages used

The code used here is dependent on these packages.

Data used

The data I am using here was obtained on 2025-May-27 from the 2021 Census - Boundary Files from Statistics Canada. I downloaded data with the following options selected: Language == English, Type == Cartographic Boundary Files (CBF), Administrative boundaries == Census subdivisions, Format == Shapefile (.shp). These data include the Census Subdivision (CSD) shapefiles used by Statistic Canada.

See the Reference Guide for Boundary Files for more information.

```
## Read in the data
Can_cities <- sf::st_read("CensusData/lcsd000b21a_e.shp")
```

```
## Reading layer 'lcsd000b21a_e' from data source
##   '/Users/nathanearley/Desktop/RetroGeoref/CensusData/lcsd000b21a_e.shp'
##   using driver 'ESRI Shapefile'
## Simple feature collection with 5161 features and 6 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 3689321 ymin: 659305 xmax: 9015751 ymax: 5242009
## Projected CRS: NAD83 / Statistics Canada Lambert
```

```
## Subset the data to only include records from BC (PRUID == "59")
BC_CSD <- Can_cities |>
  subset(PRUID == "59")
```

Generating Optimized Minimum Enclosing Circles

Optimized Minimum Enclosing Circles (OMECs) are a common point-radius method for simplifying ambiguous location data into conservative estimates of Northing, Westing, and Accuracy. OMECs are generated by creating the smallest possible circle around a polygon that includes the entire extent of the polygon. From this OMEC we can calculate a point-radius for each location where the point is the centre of the OMEC

and the radius is the radius of the OMEC. This then translates to a Northing and Westing (point) and an Accuracy (radius) that can be used in GBIF to map loaction data instead of mapping a polygon.

In some cases, the point radius is relatively simple as the point falls inside of the polygon (e.g. Figure 1).

Figure 1: Polygon of the Census Subdivision for Vancouver, BC, in blue with the OEMC in red and the centroid point in black.

In other cases the point falls outside of the polygon and the circle must therefore be adjusted so that the circle is still the smallest it can be but with the point landing on the edge of the polygon (e.g. Figure 2). The new "corrected centre" OMEC will always be larger than the "geographic centre" OMEC but the point lands within the polygon.

To calculate the OMEC and point radius for every Census Subdivision in BC I use the following code:

```
## This function does the heavy lifting for this method
compute_omec <- function(PRUID, CSDNAME, CSDTYPE, geom) {
  # Ensure geometry is an sfc object
  city <- sf::st_sf(PRUID = PRUID,
                    CSDNAME = CSDNAME,
                    CSDTYPE = CSDTYPE,
                    geometry = sf::st_sfc(geom,
                                          crs = sf::st_crs(BC_CSD)))

  # Convex hull and coordinates (used for optimization)
  hull <- sf::st_convex_hull(city)
  coords <- sf::st_coordinates(hull)[, 1:2]
```

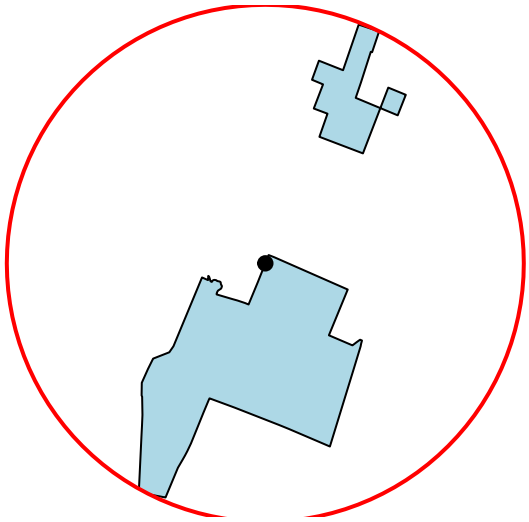

Figure 2: Polygon of the Census Subdivision for Elkford, BC, in blue with the corrected-centre OEMC in red and the centroid point in black.

```r
  # Objective function: max distance from center to hull points
  objective <- function(center) {
    max(sqrt(rowSums((coords - matrix(center,
                                      nrow = nrow(coords),
                                      ncol = 2,
                                      byrow = TRUE))^2)))
  }

  # Start point = centroid of hull
  start <- sf::st_coordinates(sf::st_centroid(hull))

  # Optimization
  opt <- optim(start, objective, method = "Nelder-Mead")
  center_utm <- opt$par
  radius <- opt$value

  # Create center point
  center_point <- sf::st_sfc(sf::st_point(center_utm),
                             crs = sf::st_crs(city))

  # Check if center is inside the original city polygon
  is_inside <- sf::st_within(center_point, city, sparse = FALSE)[1, 1]

  if (!is_inside) {
    # Move center to nearest point on polygon boundary
    nearest_on_geom <- sf::st_nearest_points(center_point, city)
    corrected_center_point <- sf::st_cast(nearest_on_geom, "POINT")[2]
    center_utm <- sf::st_coordinates(corrected_center_point)

    # Recalculate radius using all exterior coordinates of the polygon
    boundary_coords <- sf::st_coordinates(sf::st_cast(city, "MULTILINESTRING"))[, 1:2]
    radius <- max(sqrt(rowSums((boundary_coords - matrix(center_utm,
                                                         nrow = nrow(boundary_coords),
                                                         ncol = 2,
                                                         byrow = TRUE))^2)))
  }

  # Convert final center to lat/lon
  center_vect <- terra::vect(matrix(center_utm,
                                    ncol = 2),
                             crs = sf::st_crs(city)$wkt)
  center_latlon <- sf::st_as_sf(terra::project(center_vect, "EPSG:4326")) |>
    sf::st_coordinates()

  return(tibble::tibble(
    OMEC_POINT_LAT = center_latlon[2],
    OMEC_POINT_LON = center_latlon[1],
    OMEC_RADIUS_M = as.numeric(radius)
  ))
}

# Apply function across all BC_CSD
omec_results <- purrr::pmap_dfr(
```

```
  list(BC_CSD$PRUID,
       BC_CSD$CSDNAME,
       BC_CSD$CSDTYPE,
       BC_CSD$geometry),
  compute_omec
)

# Combine with original data
BC_CSD_omec <- bind_cols(BC_CSD, omec_results)
```

This code calculates the OMEC, defines the centroid, checks that the centroid is within the bounds of the polygon, and adds the latitude, longitude, and accuracy into the dataframe for each city.

Points from UBC

In which CSD is UBC included?

For some specimens, the location given may not be well encapsulated by the data we have here. In some cases, it probably isn't worth dealing with (e.g. Portage Inlet, BC can be included in View Royal without much issue) while in others, it may be totally inappropriate. The most noteworthy example here is UBC Vancouver. UBC is not included in the CSD for Vancouver but is included in Metro Vancouver A CSD (Figure 3). The corrected-centre for Metro Vancouver A is far from the UBC campus and it would be useful to use a better method.

Figure 3: Polygon of the Census Subdivision for Metro Vancouver A, BC, in grey with the UBC Campus in red and the centroid point in black.

Subset to UBC Campus

To make the UBC Campus location better we can subset the data within Metro Vancouver A to make a new row in the dataset that we'll call "UBC Campus."

```
## For the UBC Vancouver data we'd prefer to have a more specific point-radius
## So... lets build it from the Can_cities df
MetroVanA <- Can_cities |>
  subset(PRUID == "59") |>
  subset(CSDNAME == "Metro Vancouver A")
## so we just want the UBC part of MetroVanA
## To get that we'll cut out just that section
multi_geom <- st_geometry(MetroVanA)[[1]]
# length(multi_geom)  # This tells you how many separate POLYGONs it has
# Extract just the first POLYGON (index with [[n]])
single_poly <- st_sfc(st_polygon(multi_geom[[14]]),
                      crs = st_crs(MetroVanA))
# Now you can make it an sf object if needed
UBCVan <- st_sf(MetroVanA[1, -which(names(MetroVanA) == "geometry")],
                geometry = single_poly)

UBCVan$CSDNAME <- "UBC Campus"

UBC_omec <- compute_omec(
  PRUID = UBCVan$PRUID,
  CSDNAME = UBCVan$CSDNAME,
  CSDTYPE = "CUST", ## So that we know that this is a custom polygon
  geom = sf::st_geometry(UBCVan)[[1]]
)
```

```
## Warning: st_centroid assumes attributes are constant over geometries
```

```
UBCVan_with_omec <- dplyr::bind_cols(UBCVan, UBC_omec)
```

To check that the UBC Campus location is correct we'll use this code to make a figure

```
## Plot the data to check that everything works with this function
plot_omec <- function(row) {
  # 1. Extract geometry (city shape)
  city_geom <- row$geometry

  # 2. Create center point in lat/lon
  center_lonlat <- sf::st_sfc(
    sf::st_point(c(row$OMEC_POINT_LON,
                   row$OMEC_POINT_LAT)),
    crs = "EPSG:4326"
  )

  # 3. Project center to city CRS
  center_projected <- sf::st_transform(center_lonlat,
                                       sf::st_crs(city_geom))

  # 4. Create OMEC circle as buffer
  omec_circle <- sf::st_buffer(center_projected,
                               dist = row$OMEC_RADIUS_M)
```

```
  # 5. Plot city, OMEC circle, and point
  plot(city_geom,
       # main = paste("OMEC for", row$CSDNAME),
       col = "lightblue"
       )
  plot(omec_circle,
       border = "red",
       lwd = 2,
       add = TRUE)
  plot(center_projected,
       col = "black",
       pch = 19,
       add = TRUE)
}
```

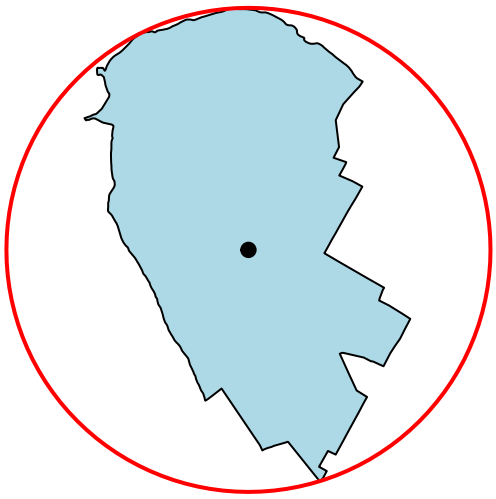

Figure 4: Polygon of the new polygon for UBC Campus in blue with the corrected-centre OEMC in red and the centroid point in black.

Add the new UBC Campus polygon into BC_CSD_omec

Now that we have this location for UBC Campus it would be useful to add it back into the BC_CSD_omec dataset for use in the future. So lets do that here:

```
## Combine them
BC_CSD_omec <- dplyr::bind_rows(BC_CSD_omec, UBCVan_with_omec)
```

```r
## Check to make sure that UBC Campus is now included
tail(BC_CSD_omec)
```

```
## Simple feature collection with 6 features and 9 fields
## Geometry type: GEOMETRY
## Dimension:     XY
## Bounding box:  xmin: 4008419 ymin: 2005295 xmax: 4723366 ymax: 3167136
## Projected CRS: NAD83 / Statistics Canada Lambert
##      CSDUID            DGUID          CSDNAME CSDTYPE   LANDAREA PRUID
## 747 5959007 2021A00055959007 Northern Rockies     RGM 84759.3073    59
## 748 5959805 2021A00055959805         Fontas 1     IRI     0.1287    59
## 749 5959806 2021A00055959806    Fort Nelson 2     IRI    95.5757    59
## 750 5959809 2021A00055959809        Kahntah 3     IRI     0.0825    59
## 751 5959810 2021A00055959810  Prophet River 4     IRI     3.7905    59
## 752 5915020 2021A00055915020       UBC Campus     RDA   815.2061    59
##     OMEC_POINT_LAT OMEC_POINT_LON OMEC_RADIUS_M                       geometry
## 747       59.05961      -123.7448   241549.5473 MULTIPOLYGON (((4357079 316...
## 748       58.28539      -121.7262      275.0454 MULTIPOLYGON (((4555464 284...
## 749       58.79540      -122.5493     8958.1539 MULTIPOLYGON (((4540228 291...
## 750       58.35594      -120.9077      230.9426 MULTIPOLYGON (((4600328 282...
## 751       58.08022      -122.6950     1681.3473 MULTIPOLYGON (((4496803 284...
## 752       49.25470      -123.2275     3106.3925 POLYGON ((4010352 2011359, ...
```

Conclusion

Using the code above we should be able to tackle most issues relating to retrospective georeferencing in a systematic way, at least when the collection data includes a reasonable reference to a Canadian Census Subdivision. In cases where a location can be easily extracted from a MULTIPOLYGON included in the Canadian Census Subdivisions (like the UBC example). I've included code for that.

Edge Cases

Sometimes collection data is too specific to follow this systematic approach.

In cases when it is preferable to sacrifice collection site specificity in order to maintain systematic methodology — when a polygon can not be easily obtained — the specific collection site can be listed in collection data notes while using the point-radius data of the Canadian Census Subdivision that is most appropriate to the location (e.g. Fernwood Neighbourhood -> Victoria, BC).

In cases when it is preferable to maintain collection site specificity sacrifice the systematic methodology — when a polygon can not be easily obtained — one can assign a point near the centre of the collection site listed and assign an accuracy radius that includes the whole extent of the location. When this method is employed it is inappropriate to list Zermoglio et al. (2020) as the protocol used and the details of this method should be explained wherever possible.