Donaldson and Hornbeck (2016) Network replication

Thursday, November 8, 2018

1. Download the `Donaldson_2016_rep.zip` file from the course website. Start a new script, load usual packages (`tidyverse`, `sf`, and `units`) and the new `dodgr` package. Set your working directory.

2. Open the county data (`./dh_shapes/US_county_1890.shp`). Subset the data to counties in Pennsylvania (via `STATENAM`). What projection are the data in? Get them into EPSG:4326. Do the same with the centroids (`./dh_shapes/Centroids_1890.shp`). Plot the counties with their centroids to reach replication goal 1.

3. DH (2016) create direct wagon routes between the county centroids. Use `st_nearest_points()` to find these routes, `st_sf()` to create a data frame, `st_length()` to find the line length, and then subset to all lines shorter than 300 km. Be careful with unit comparisons. Plot the routes to verify your computations. Unselect all variables and create a new one called `waytype` which is set to `"wagon"` for all observations.

4. Load the railway lines in 1870 (`./dh_shapes/rail_1870.shp`), unselect all variables and then add a new variable called `waytype` which is always set to `"railway"`.

5. DH (2016) connect every county's centroid to the nearest railway line via a wagon route (i.e. a straight line). Use `st_nearest_feature()` to identify the nearest line and then `st_nearest_points()` to connect the county centroids only to the nearest line (you'll need to subset to the nearest index and use `pairwise=T`). Turn it into a simple features data frame and add a `waytype` variable which is always set to `"railaccess"`. Plot the railways in 1870 and your access routes to reach replication goal 2.

6. Use `rbind()` to tie all three components of the network together. Now you need to make sure everything is really precisely connected. Take the entire network and project it into a meaningful CRS with meters as native units. Snap the vertices of the network to itself using a tolerance of 50 m. You need to make sure that your centroids also exactly fit to this slightly modified network, so project them as well and snap them to the network.[1] Transform both back to the geograpic projection and give each element in the network a unique ID.

6. Make a table of the different way types in the network. Then use `dodgr`'s `weight_streetnet()` to convert the vectors into a network. Provide your ID to `id_col=`, use `type_col = "waytype"` and set the weighting profile to `wt_profile=rep(1,3)`. Careful the last step does not create the appropriate weights, we will modify them shortly. Examine your graph version of the network, esp. the variable `d_weighted`.[2]

7. Use the following lines to create the correct weights with Fogel's (1964) parameters and then reassign these weights to the distance variable. The latter step is only necessary because DH (2016) are interested in the accumulated dollar cost of each path, not the length of each path.

```
# do not use tidy style, will kill dodgr class
my.net$d_weighted[my.net$highway == "wagon"] <-
  0.231*my.net$d[my.net$highway == "wagon"]/1.609344
my.net$d_weighted[my.net$highway == "railway"] <-
  0.00627324613*my.net$d[my.net$highway == "railway"]/1.609344
my.net$d_weighted[my.net$highway == "railaccess"] <-
  0.231*my.net$d[my.net$highway == "railaccess"]/1.609344 + 0.5
my.net$d <- my.net$d_weighted
```

[1] This step was necessary in an earlier version of this exercise, where my `dodgr_route_sf()` function was not able to start at nodes that are close but not directly on the network. `dodgr_dists()` is a bit more forgiving in this respect and allows for small discrepancies in the coordinates. My function now has an argument called `snapdist` which allows for (small) snapping distance and is set in meters. You probably want to leave the argument empty. Then it will default to snap all coordinates within 1 m to the next node on the network.

[2] We had major issues in class with `weight_streetnet()` not working on some computers. It turns out that `R 3.5.X` changed how the `duplicated()` function handles data frames. This could be intended or a major bug in base R. Time will tell. If you have `R 3.5.X`, then please use: `if (R.Version()$major == "3" & as.numeric(R.Version()$minor) > 4.4) source("./weight_streetnet_rb.R")`, which modifies the `duplicated()` call inside `weight_streetnet()` and makes it behave just like in `R 3.4.4` or earlier. Now the script should run on all computers.

```
head(my.net)
```

8. Convert your county centroids to a data frame (use `st_coordinates()` first). Then use `dodgr_dists()` to compute the cost matrix $\boldsymbol{\tau}$. You may set `parallel = T` to speed things up. Take a look at the first five rows and columns to reach replication goal 3.

9. To visualize the shortest routes you will have to load my function `source("./dodgr_route_sf.R")` and then use `dodgr_route_sf()` to calculate all shortest paths from one place to all others. It takes your `sf`-object of the county coordinates as an argument and otherwise works like `dodgr_dists()`. With `quiet=F` it will talk to you. You can now reach goal 4 and goal 5 if you run verything with the 1890 railways.