Many-to-Many Point Computation Script

Luka Vukovic

Loading libraries

```
# Main
library(r5r)
library(sf)
library(data.table)

if (Sys.getenv("JAVA_HOME")!="") { Sys.setenv(JAVA_HOME="") }
library(rJava)

# Convenience
library(tidyverse)
library(glue)
library(rlist)

# For pretty knitting
library(lemon)
knit_print.data.frame <- lemon_print
knit_print.tbl <- lemon_print
knit_print.summary <- lemon_print

path_data <- "../../data/2_clean/"
```

1. Setting Up the Network Graph

- To compute transit accessibility measures we must first build the transit network in R using the r5r library.
- If the vancouver_canada.osm.pbf file needs to be converted to an .osm, one can use a binary osm converter available at: https://wiki.openstreetmap.org/wiki/Osmconvert#Binaries

Note: Graph network building may take up to a few minutes, especially for large cities

```
# allocate an appropriate RAM to Java
options(java.parameters = "-Xmx6g")

# build the transit network
# OSM file and GTFS.zip are in the same directory in this case
r5r_core <- setup_r5(data_path = getwd(), verbose = FALSE)
```

2. Loading the Origins and Destinations

- Origins correspond to the centroid coordinates of Canadian Census 2016 dissemination blocks across Canada. This was filtered by metropolitan area (MA) to keep only Greater Vancouver city blocks.
- Destinations correspond to the amenity coordinates of the OCDAF database.

```r
## ORIGINS

# dissemination blocks via fread, a faster way to import than read_csv
origins <- fread(file.path(paste0(path_data, "vancouver_db.csv")))

# remove population column
origins <- origins[, -2]

# conver id numeric to char
origins$id <- as.character(origins$id)

paste('Origins: ', nrow(origins))
```

```
## [1] "Origins:  15197"
```

```r
head(origins)
```

```
##             id     lat       lon
## 1: 59150004004 49.3739 -123.2738
## 2: 59150004005 49.3746 -123.2757
## 3: 59150004006 49.3738 -123.2763
## 4: 59150004011 49.3735 -123.2725
## 5: 59150004012 49.3725 -123.2729
## 6: 59150004013 49.3724 -123.2728
```

```r
## DESTINATIONS

# cultural/Art facilities
destinations <- fread(file.path(paste0(path_data, "vancouver_amenities.csv")))

# see summary counts of each amenity
destinations %>% group_by(type) %>% summarise(count = n()) %>% arrange(desc(count))
```

type	count
gallery	99
museum	92
library or archives	88
theatre/performance and concert hall	75
artist	48
heritage or historic site	28
miscellaneous	6
art or cultural centre	5
festival site	2

```r
# filter amenity types to keep the amenities of interest
target_amenities <- c('gallery',
                      'museum',
                      'library or archives',
                      'theatre/performance and concert hall')

destinations <- destinations %>% filter(type %in% target_amenities)

# keep id, lat, and lon columns
destinations <- destinations[, 1:3]
```

```r
destinations$lat <-  as.numeric(destinations$lat) # char to numeric
destinations$lon <-  as.numeric(destinations$lon) # char to numeric
destinations$id <- as.character(destinations$id)  # numeric to char

paste('Destinations with NA: ', nrow(destinations))
```

```
## [1] "Destinations with NA:  354"
```

```r
destinations <- destinations[complete.cases(destinations)] # remove NA rows
paste('Destinations clean: ', nrow(destinations))
```

```
## [1] "Destinations clean:  346"
```

```r
# peek
head(destinations)
```

```
##     id      lat       lon
## 1:  10 49.17635 -123.1128
## 2:  15 49.26194 -123.1511
## 3:  24 49.27879 -123.0988
## 4: 115 49.27922 -123.1162
## 5: 157 49.26391 -123.2549
## 6: 215 49.05554 -122.4819
```

3. Setting Travel Constraints

- For each computed travel time via transit, we set constraints to model more realistic uses of transit networks.
- For example, most people wont take the bus if it take longer than 2 hours, or if they need to walk more than a kilometer on the trip, or if they need to take more than 3 transfers and so on.

```r
# Non-transit modes: WALK, BICYCLE, CAR, BICYCLE_RENT, CAR_PARK
# Transit modes: TRAM, SUBWAY, RAIL, BUS, FERRY, CABLE_CAR, GONDOLA, FUNICULAR
# default walk speed = 3.6 km/h

mode <- c('WALK', 'TRANSIT')
max_walk_dist <- 1000 # 1 km
max_trip_duration <- 120 # 2 hours
max_rides <- 3 # max transfers
```

4. Computing the Travel Time Matrix

- To measure a city blocks accessibility to all amenities within the constraints,we need to consider averaging travel times across both a weekly bus schedule, a saturday bus schedule, and a sunday bus schedule.
- Furthermore, we also want to average transit times across the time of day so we compute a travel time every hour from 7am to 7pm with a departure window of 30 minutes.

Note: this operation only functions on a single cpu core so the cell can take a few hours to execute. Please be patient or consider using less origins or destinations or time points.

```r
# collect each travel time matrix from every hour
all_ttms <- list()

# computing for May 14/15/16 (Fri/Sat/Sun)
for (day in 14:16) {
  # computing from 7am to 7pm
```

```
  for (time in 7:19) {

    departure_datetime <- as.POSIXct(glue("{day}-05-2021 {time}:00:00"), format="%d-%m-%Y %H:%M:%S")

    ttm <- travel_time_matrix(r5r_core = r5r_core,
                              origins = origins,
                              destinations = destinations,
                              departure_datetime = departure_datetime,
                              time_window = 30,

                              # constraints
                              mode = mode,
                              max_walk_dist = max_walk_dist,
                              max_trip_duration = max_trip_duration,
                              max_rides = max_rides,
                              verbose = FALSE)

    # do NOT use rbind(all_ttm, ttm), it is insultingly slow
    # append to a list then use rbindlist
    all_ttms <-  list.append(all_ttms, ttm)

    print(glue('Progress: {round(((day-14)*12 + time-6)/37*100, 1)}%'))

  }
}

# Fast way to bind all data.tables
TTM <- rbindlist(all_ttms)

print('COMPLETED')

summary(TTM)
```

5. Aggregating Travel Time Matrix

- We are interested in the average transit time across all 36 departure times so we aggregate on the origin and destination (ie. on every unique trip).
- For comparing transit accessibility between different days and time see the **ttm_time_unaggregated** folder.

```
TTM_agg <- TTM %>%
           group_by(fromId, toId) %>%
           summarise(
             avg_time = mean(travel_time), # average time
             sd_time = sd(travel_time)     # standard deviation of avg time
           )
```

```
# for knitting we just skip the TTM computation and aggregation
TTM_agg <- read.table(
                 unz("../../data/3_computed/main_travel_time_matrix--time_aggregated.zip",
                     "main_travel_time_matrix--time_aggregated.csv"), # file within zip
                  header=T, quote="\"", sep=",") # format into a table

paste('"First" aggregation:')
```

```
## [1] "\"First\" aggregation:"
```

```
paste('Note this is the cleaned version that was imported to skip the computation cell during knitting.
```

```
## [1] "Note this is the cleaned version that was imported to skip the computation cell during knitting
```

```
summary(TTM_agg)
```

```
##      fromId              toId         avg_time         sd_time
##  Min.   :5.915e+10   Min.   :  10   Min.   :  0.00   Min.   : 1.000
##  1st Qu.:5.915e+10   1st Qu.:3135   1st Qu.: 50.59   1st Qu.: 1.847
##  Median :5.915e+10   Median :5572   Median : 70.95   Median : 2.719
##  Mean   :5.915e+10   Mean   :5503   Mean   : 71.70   Mean   : 3.235
##  3rd Qu.:5.915e+10   3rd Qu.:8182   3rd Qu.: 93.46   3rd Qu.: 4.017
##  Max.   :5.915e+10   Max.   :9780   Max.   :119.00   Max.   :35.355
```

6. Fixing Odd Values in sd_time

NA standard deviations - Since some origins may only have only a single trip, their standard deviation will be undefined as you need at least 2 values for standard deviation. - We simply replace them with the median trip standard deviation. - This is not the worst assumption to make since the bus still probably comes on a regular schedule with a standard uncertainty to most stops.

Zero standard deviations - We can also imagine many trips have no standard deviation as their travel time is always identical across multiple days or hours. - To avoid any issues with zero division or massive numerators from small division (in case we use a different scoring formula), we can replace these cases with 1 minute since 1 minute is the smallest realistic standard deviation in any travel time.

```
# replace NAs
median_sd <- median(TTM_agg$sd_time, na.rm=TRUE)
TTM_agg <- TTM_agg %>% replace_na(list('sd_time' = median_sd))

# replace < 1 with 1 to avoid large or infinity computations later on
TTM_agg$sd_time[(TTM_agg$sd_time < 1)] <-  1

paste('Clean TTM Aggregation:')
```

```
## [1] "Clean TTM Aggregation:"
```

```
summary(TTM_agg)
```

```
##      fromId              toId         avg_time         sd_time
##  Min.   :5.915e+10   Min.   :  10   Min.   :  0.00   Min.   : 1.000
##  1st Qu.:5.915e+10   1st Qu.:3135   1st Qu.: 50.59   1st Qu.: 1.847
##  Median :5.915e+10   Median :5572   Median : 70.95   Median : 2.719
##  Mean   :5.915e+10   Mean   :5503   Mean   : 71.70   Mean   : 3.235
##  3rd Qu.:5.915e+10   3rd Qu.:8182   3rd Qu.: 93.46   3rd Qu.: 4.017
##  Max.   :5.915e+10   Max.   :9780   Max.   :119.00   Max.   :35.355
```

7. Exporting the Travel Time Matrix for Further Work

- This summarizes our primary method for the first step in efficient transit network analysis: **Obtaining Realistic Travel Time Data**

```
# for compressed output - otherwise file is too big to be pushed to github
library('readr')

write_csv(TTM_agg, "../../data/3_computed/main_travel_time_matrix--time_aggregated.csv.gz")
```

```
# you may use this function to perform imports on compressed files:
# data_here <- read.table(
#                 unz("path/to/data/3_computed/main_travel_time_matrix--time_aggregated.zip",
#                     "main_travel_time_matrix--time_aggregated.csv"), # file within zip
#                 header=T, quote="\"", sep=",") # format into a table
```