Traffic flow analysis from Uber movement data

Roman Portela Suarez & Sergio Sancho Sanz

Contents

Introduction

```
adj_matrix <- gTouches(polys, byid = TRUE)
```

```
plot(polys, main = "Zones in Amsterdam", col="#f44336",border="black", lwd=2)
```

Creating the graphs

```
#Template graph: undirected and weighted (value 1 by default). Some centralities may fail for weight=0
g_original <- graph_from_adjacency_matrix(adj_matrix, weighted=TRUE, mode="undirected") %>% set_edge_at

#Calculate all shortest paths
sp <- list()
for (i in 1:nrow(adj_matrix)){
  sp[[length(sp) + 1]] <- get.shortest.paths(g_original, i)
}

#g is a list of graphs, one per hour of day
g <- list()

#Centralities:
degree.cent <- list()
```

```r
degree.str.cent <- list()
closeness.cent <- list()
betweenness.cent <- list()
eigen.cent <- list()
page.rank.cent <- list()

#Communities
betweenness.com <- list()
fast.greedy.com <- list()
label.prop.com <- list()
spinglass.com <- list()
walktrap.com <- list()
optimal.com <- list()

#Index i=TRUE means that graph i is weakly connected
is_connected <- list()
```

```r
for (hour in 1:24){
  current_data <- subset(amsterdam, hod==hour-1)
  g[[length(g) + 1]] <- g_original

  for(i in 1:nrow(current_data)) {
    trip <- current_data[i,]
    #ID = position in matrix
    from <- trip$sourceid
    to <- trip$dstid

    #Se convierte el camino A B C en vector A B B C
    route <- as.vector(sp[[from]]$vpath[[to]])
    EP = rep(route, each=2)[-1]
    EP = EP[-length(EP)]

    #Coger ID de arista AB y BC (para acceder a E(g[[hour]]))
    edge_ids <- get.edge.ids(g[[hour]], EP)

    #Incrementar weight de arista AB y BC
    E(g[[hour]])$weight[edge_ids] = E(g[[hour]])$weight[edge_ids] + 1

  }

  is_connected[[length(is_connected) +1]] <- is.connected(g[[hour]], mode="weak")

  ##CENTRALITIES
  #Grado de un nodo (sin peso)
  degree.cent[[length(degree.cent) + 1]] <- centr_degree(g[[hour]], mode = "all")

  #Strength = suma de pesos de aristas por nodo
  degree.str.cent[[length(degree.str.cent) + 1]] <- strength(g[[hour]], mode = "all")

  #Numero de caminos con menos tráfico que pasan por un nodo (suma de pesos minimizada)
  betweenness.cent[[length(betweenness.cent) + 1]] <- betweenness(g[[hour]], directed = FALSE)

  #Parecido a pagerank. Una zona está congestionada si lo está, o si sus adyacentes también. Cuanto may
```

```
  eigen.cent[[length(eigen.cent) + 1]] <- eigen_centrality(g[[hour]], directed = FALSE)

  #This is a disconnected graph, so closeness may not be usefull
  closeness.cent[[length(closeness.cent) + 1]] <- closeness(g[[hour]], mode = "all")

  #PageRank
  page.rank.cent[[length(page.rank.cent)+1]] <- page_rank(g[[hour]])



  ##COMUNITIES
  betweenness.com[[length(betweenness.com) + 1]] <- cluster_edge_betweenness(g[[hour]])
  fast.greedy.com[[length(fast.greedy.com) + 1]] <- cluster_fast_greedy(g[[hour]])
  label.prop.com[[length(label.prop.com) + 1]] <- cluster_label_prop(g[[hour]])
  walktrap.com[[length(walktrap.com) + 1]] <- cluster_walktrap(g[[hour]])

}
```

Analysis

```
#Number of connected graphs
sum(as.integer(is_connected))
```

```
#Show centralities
rbPal <- colorRampPalette(c('#ffcdd2','#b71c1c'))

#Strength
png(filename="img\\degree.str.cent\\%02d.png")
for (hour in 1:24){
  Colores <- rbPal(100)[as.numeric(cut(degree.str.cent[[hour]],breaks = 100))]
  plot(polys, main = paste("Strength degree centrality in Amsterdam at ", (hour-1), ":00 h", sep=""), co
}
dev.off()
system("\"C:\\Program Files\\ImageMagick-7.0.8-Q16\\magick.exe\" -delay 80 img\\degree.str.cent\\*.png i
#file.remove(list.files(pattern=".png"))


#Degree
png(filename="img\\degree.cent\\%02d.png")
for (hour in 1:24){
  Colores <- rbPal(100)[as.numeric(cut(degree.cent[[hour]]$res,breaks = 100))]
  plot(polys, main = paste("Degree centrality in Amsterdam at ", (hour-1), ":00 h", sep=""), col=Colores
}
dev.off()
system("\"C:\\Program Files\\ImageMagick-7.0.8-Q16\\magick.exe\" -delay 80 img\\degree.cent\\*.png img\
#file.remove(list.files(pattern=".png"))


#Closeness
png(filename="img\\closeness.cent\\%02d.png")
for (hour in 1:24){
```

```
  Colores <- rbPal(100)[as.numeric(cut(closeness.cent[[hour]],breaks = 100))]
  plot(polys, main = paste("Closeness centrality in Amsterdam at ", (hour-1), ":00 h", sep=""), col=Col
}
dev.off()
system("\"C:\\Program Files\\ImageMagick-7.0.8-Q16\\magick.exe\" -delay 80 img\\closeness.cent\\*.png i
#file.remove(list.files(pattern=".png"))


#Betweenness
png(filename="img\\betweenness.cent\\%02d.png")
for (hour in 1:24){
  Colores <- rbPal(100)[as.numeric(cut(betweenness.cent[[hour]],breaks = 100))]
  plot(polys, main = paste("Betweenness centrality in Amsterdam at ", (hour-1), ":00 h", sep=""), col=C
}
dev.off()
system("\"C:\\Program Files\\ImageMagick-7.0.8-Q16\\magick.exe\" -delay 80 img\\betweenness.cent\\*.png
#file.remove(list.files(pattern=".png"))


#Eigen
png(filename="img\\eigen.cent\\%02d.png")
for (hour in 1:24){
  Colores <- rbPal(100)[as.numeric(cut(eigen.cent[[hour]]$vector,breaks = 100))]
  plot(polys, main = paste("Eigen centrality in Amsterdam at ", (hour-1), ":00 h", sep=""), col=Colores
}
dev.off()
system("\"C:\\Program Files\\ImageMagick-7.0.8-Q16\\magick.exe\" -delay 80 img\\eigen.cent\\*.png img\\
#file.remove(list.files(pattern=".png"))


#PageRank
png(filename="img\\page.rank.cent\\%02d.png")
for (hour in 1:24){
  Colores <- rbPal(100)[as.numeric(cut(page.rank.cent[[hour]]$vector,breaks = 100))]
  plot(polys, main = paste("PageRank centrality in Amsterdam at ", (hour-1), ":00 h", sep=""), col=Colo
}
dev.off()
system("\"C:\\Program Files\\ImageMagick-7.0.8-Q16\\magick.exe\" -delay 80 img\\page.rank.cent\\*.png i
#file.remove(list.files(pattern=".png"))
```

Color comunidades

```
png(filename="img\\betweenness.com\\%02d.png")
for (hour in 1:24){
  plot(polys, main = paste("Communities in Amsterdam at ", (hour-1), ":00 h", sep=""), col=membership(b
}
dev.off()
system("\"C:\\Program Files\\ImageMagick-7.0.8-Q16\\magick.exe\" -delay 80 img\\betweenness.com\\*.png


png(filename="img\\fast.greedy.com\\%02d.png")
```

```
for (hour in 1:24){
  plot(polys, main = paste("Communities in Amsterdam at ", (hour-1), ":00 h", sep=""), col=membership(fa
}
dev.off()
system("\"C:\\Program Files\\ImageMagick-7.0.8-Q16\\magick.exe\" -delay 80 img\\fast.greedy.com\\*.png i


png(filename="img\\label.prop.com\\%02d.png")
for (hour in 1:24){
  plot(polys, main = paste("Communities in Amsterdam at ", (hour-1), ":00 h", sep=""), col=membership(la
}
dev.off()
system("\"C:\\Program Files\\ImageMagick-7.0.8-Q16\\magick.exe\" -delay 80 img\\label.prop.com\\*.png im


png(filename="img\\walktrap.com\\%02d.png")
for (hour in 1:24){
  plot(polys, main = paste("Communities in Amsterdam at ", (hour-1), ":00 h", sep=""), col=membership(wa
}
dev.off()
system("\"C:\\Program Files\\ImageMagick-7.0.8-Q16\\magick.exe\" -delay 80 img\\walktrap.com\\*.png img
```

Introduction (R)

Travelling from one place to another is a hard task in big cities. For that reason, some companies like Uber provide their users with cars, so they can easily travel in exchange of some money. Also, users can take the role of the drivers and get a payment. This service is specially important for tourists: they do not own a vehicle, and using the public transport may take a long time. There are many advantages and disadvantages about services of this kind, but it is not the main objective of this document to discuss them.

Uber provides some datasets related with the use of their services in different cities around the world. It is possible to use that data in order to create a graph, where city areas are the nodes and their adjacencies are the edges. Next, we can analyze the traffic flow between different parts of a city. By doing that, it is the objective of this document to detect congested areas (meaning areas where the Uber traffic flow is very dense). Once that metric is calculated, it would be possible to use different criteria to find out the most congested area. That would mean that it is the area where most cars drive in and out, and so it is an important area. Moreover, some community detection algorithms can also be applied to this scenario. By performing this type of analysis, it is possible to identify groups of congested areas (or, in the same way, non congested areas).

Those results could be used to identify which route can be followed to avoid dense traffic, or to identify important parts of the city (assuming that cars tend to drive in important areas).

Once different criteria are used for each kind of annalysis, some discussion will be made regarding how robust or accurate are the used method, together with some possible future improvements.

Dataset (R)

Que datos tenemos : info, formato, fuentes, qué datos interesan, viajes vs zonas. Explicar corto GeoJSON. Problemas: no tenemos rutas, solo origen y destino.

We can find a huge amount of information in the Uber website. Once there, we can get datasets which gather information on different big cities around the world, such as London, Paris, Amsterdam, Chicago, Los Angeles, New Dheli or Sydney. The only difference between datasets of different cities is the amount of data, because all of them are characterized by the same information. We chose to analyze the traffic flow in Amsterdam. Hence, every piece of information from this point on was analyzed for this city.

There are two types of files in the datasets: geographical data (a GeoJSON file), and trips data (a CSV file).

GeoJSON is a format used for describing geographical data using JSON. It allows to define many different areas, shapes or others, using spatial objects (points, lines, polygons. . .) to do so. These objects are created from points, which are located by coordinates. In the case of the Uber dataset, GeoJSON is used to define areas of a city. An area is defined by a polygon, which can belong to one of the three cathegories below.

- **Simple polygon:** the area is enclosed into an ordinary polygon.
- **Simple polygon with holes:** the area is enclosed into a simple polygon, but there are parts inside that polygon which do not belong to the original area. In this case, The area is represented as an array of polygons. The first one defines the outer ring, while the rest of them define the holes.
- **Multipolygon:** the area is formed by different islands. In this case, it is represented by several polygons, each one of them corresponding to one island.

Each area is a polygon, and is identified by an ID. In the case of Amsterdam, there are 181 of them, where the IDs are numbers from 0 to 180. Since GeoJSON is a well kown format, there are different tools to read data from the previous file. The code used for that purpose is shown below, together with the geographical representation of the zones in Amsterdam. Recall that a zone can be formed by different polygons, or have

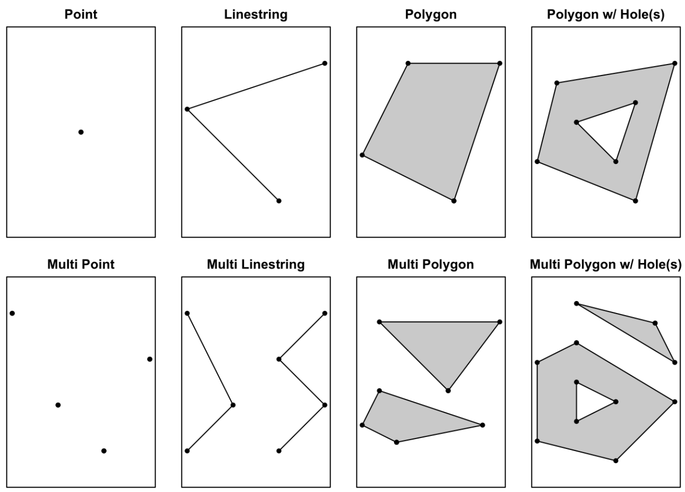

Figure 1: Different types of spatial objects, used in GeoJSON

holes. However, the vast majority of zones are defined by simple polygons, and those cases could barely be seen. Hence, all of them are shown using the same color in the figure below.

```
polys <- readOGR("data\\amsterdam_wijk.json", verbose=FALSE)
plot(polys, main = "Zones in Amsterdam", col="#f44336",border="black", lwd=2)
```

The other type of file in the dataset is the Uber related one. It is a CSV file where each row corresponds to a single trip. A trip has an origin and a destination, it occured at a time and it has a duration, represented by several statisticals. An example of this kind of data can be found below.

```
kable(amsterdam[1:5,1:4], caption = "Uber data (columns 1 to 4)")
```

Table 1: Uber data (columns 1 to 4)

sourceid	dstid	hod	mean_travel_time
143	141	7	258.90
132	12	7	662.86
130	32	7	584.99
8	173	11	870.56
172	176	10	289.16

```
kable(amsterdam[1:5,5:7], caption = "Uber data (columns 5 to 7)")
```

Table 2: Uber data (columns 5 to 7)

standard_deviation_travel_time	geometric_mean_travel_time	geometric_standard_deviation_travel_time
213.93	205.88	2.09
283.84	623.17	1.38
306.62	511.66	1.68
407.70	807.46	1.42
216.32	229.97	1.97

As seen above, the dataset only provides the origin and destination zone ID of the trip, and we do not know the route followed by the car.

Graph (S)

Como se ha creado el grafo: con pesos, no dirigido. Por que? Pesos -> trafico No dirigido: el hecho de hacerlo dirigido indica que un coche entra o sale de la zona. Sin embargo, al ser la zona un conjunto de carreteras, de las cuales no conocemos su trayectoria ni sentido (es decir, la zona es una BLACK BOX), no podemos saber si una calle en concreto está congestionada SOLO en un sentido.

Talk about how we implement the shortest path between source and destiny (shortest path in graph). Cosas que no pudimos hacer: rutas por calles con API

Centrality measures (R)

Explicar distintas medidas. Referenciar gifs. Poner algunas imagenes de las centralidades.

Communities detection (S)

Talk about spinglass and optimal communities detection. They could not be used for detecting communities since spinglass does not work with unconnected graphs and optimal algorithm takes a lot of time and resources.

Conclusion and future work (R/S)

IR AÑADIENDO SEGUN SURJA

Problemas: -El punto de partida no es por coordenadas, sino por ID de zona. Para zonas grandes, si se quiere usar Maps, esto es un problema. -Las zonas grandes son más propensas a ser elegidas como ruta más corta, a pesar de que geográficamente son grandes (en el grafo solo es un nodo) -Pros de SP por nodos: simple, rápido (si pocos nodos) -Cons de SP por nodos: Esta ruta no se tiene por que corresponder con la real.

Mejoras: -Cómo mejorar (usar Maps o similares), calcular optimal (que tarda mucho). Tratar