Parcial2-TDA2

May 20, 2022

[ ]:
```
import os, sys

from google.colab import drive
drive.mount('/content/gdrive')
```

[ ]:
```
#load the module in  a  new notebook
import sys
sys.path.append('/content/gdrive/My Drive/social-networks-utils-main\motifs')
```

[ ]:
```
import sys
sys.path.insert(0,'/content/drive/My Drive/social-networks-utils')
```

1 Resolución del parcial

Aclaración: las preguntas a desarrollar no fueron respondidas en el notebook sino en el informe realizado.

1. a. Obtener una visualización de las comunidades presentes en dicha red (indicando el algoritmo utilizado).

En primer lugar volvemos a generar el grafo correspondiente a los datos de vuelos dada en el enunciado

[ ]:
```
import networkx as nx
```

[ ]:
```
import pandas as pd
```

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
df = pd.read_csv('World.csv')
G = nx.from_pandas_edgelist(df, 'Origen', 'Destino')
```

Una vez generado el grafo. Proseguimos a realizar la visualización de las comunidades presentes en la red

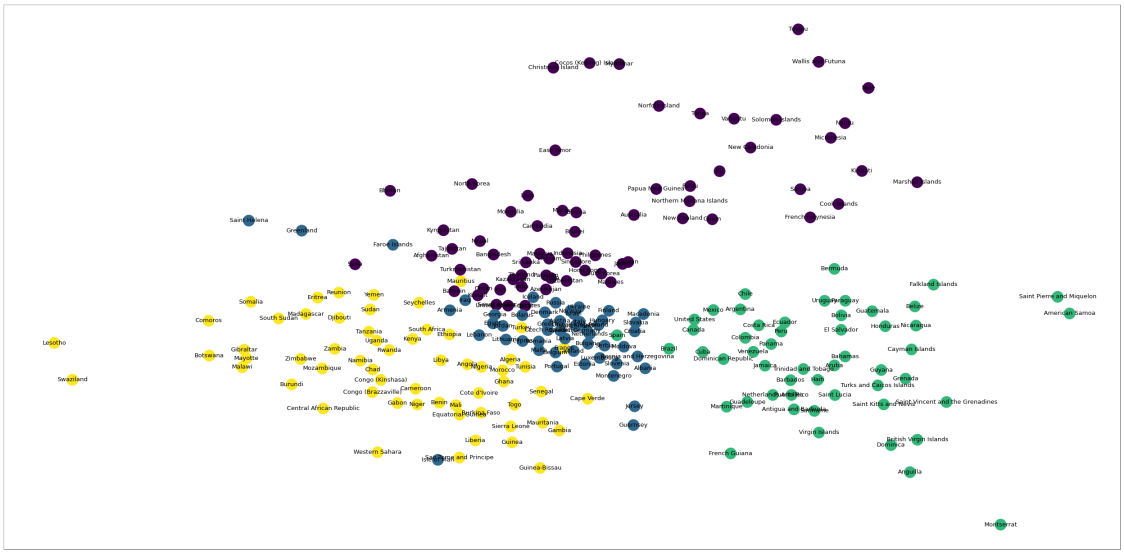
Se elige el Algoritmo de Louvain: Es un algoritmo Greedy que ejecuta en O(E log V).

```
[ ]: from community import community_louvain
```

```
[ ]: comms = community_louvain.best_partition(G)
```

Realizamos la visualización pedida:

```
[ ]: from community import community_louvain
     import matplotlib.cm as cm
     import matplotlib.pyplot as plt
     plt.figure(figsize = (40,20), dpi=80)
     # draw the graph
     pos = nx.spring_layout(G)
     # color the nodes according to their partition
     cmap = cm.get_cmap('viridis', max(comms.values()) + 1)
     nx.draw_networkx_nodes(G, pos, comms.keys(), node_size=500,
                            cmap=cmap, node_color=list(comms.values()))
     labels = nx.draw_networkx_labels(G,pos)
     #nx.draw_networkx_edges(G, pos, alpha=0.5)
     plt.show()
```



1. c. Obtener los nodos correspondientes a una de las subredes (con al menos 20% de los nodos), y realizar una visualización de las sub-comunidades presentes.

El 20% de 225 (cantidad de nodos totales) es 45. En primer lugar visualizo qué cantidad de nodos hay para cada comunidad y a partir de allí elijo alguna que cumpla con el requisito

[ ]:
```
s = pd.Series(comms)
s.value_counts()
```

[ ]:
```
0    66
1    60
3    52
2    51
dtype: int64
```

Dados los resultados anteriores, podemos seleccionar cualquiera de las comunidades presentes. Vamos elegir la opción de la comunidad 2.

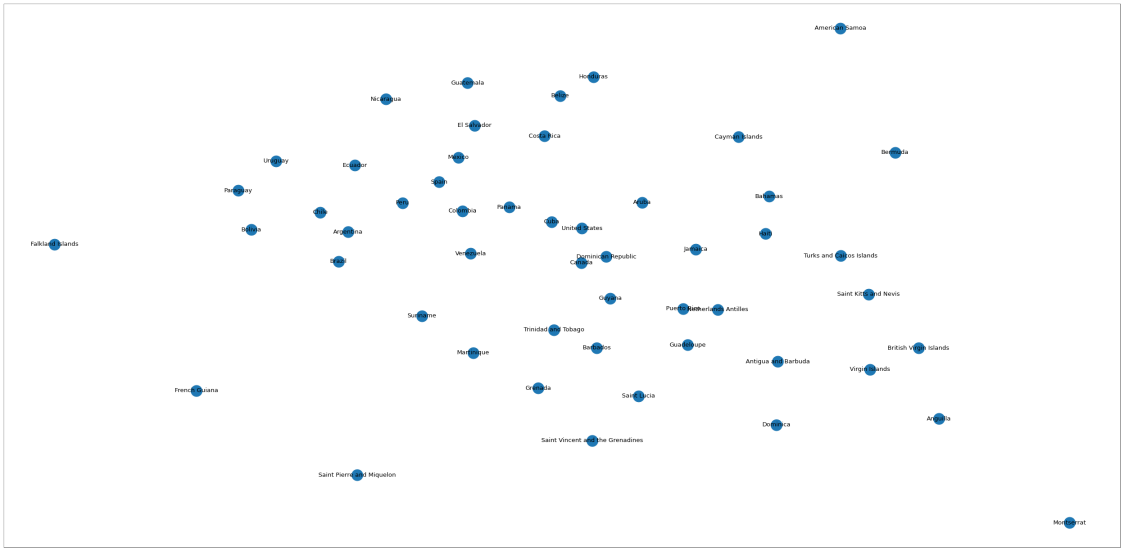
[ ]:
```
comm2 = dict(filter(lambda x: x[1] == 2,comms.items()))
comm2
```

Obtenemos el subgrafo

[ ]:
```
sub_G = G.subgraph(comm2.keys())
```

Visualización del subgrafo calculado anteriormente

[ ]:
```
from community import community_louvain
import matplotlib.cm as cm
import matplotlib.pyplot as plt
plt.figure(figsize = (40,20), dpi=80)
# draw the graph
pos_2 = nx.spring_layout(sub_G)
# color the nodes according to their partition
nx.draw_networkx_nodes(sub_G, pos_2, comm2.keys(), node_size=500)
labels = nx.draw_networkx_labels(sub_G,pos_2)
#nx.draw_networkx_edges(G, pos, alpha=0.5)
plt.show()
```

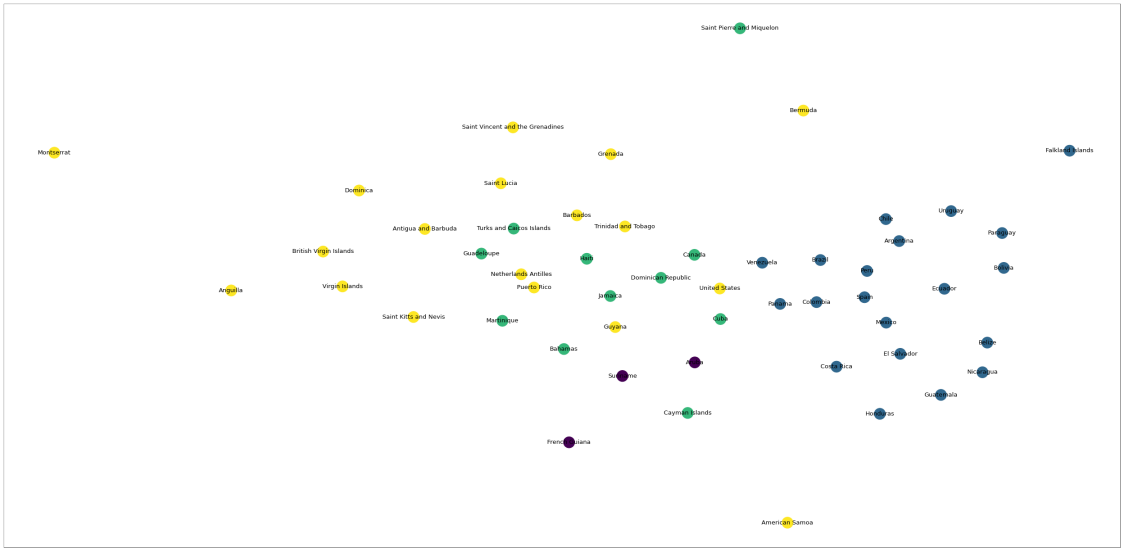


Nuevamente utilizando el Algoritmo de Louvain procedemos a calcular las subcomuniades presenten en el subgrafo calculado previamente.

[ ]:
```
subcomms = community_louvain.best_partition(sub_G)
subcomms
```

Visualización de las sub-comunidades presentes

[ ]:
```
from community import community_louvain
import matplotlib.cm as cm
import matplotlib.pyplot as plt
plt.figure(figsize = (40,20), dpi=80)
# draw the graph
pos_2 = nx.spring_layout(sub_G)
# color the nodes according to their partition
cmap = cm.get_cmap('viridis', max(subcomms.values()) + 1)
nx.draw_networkx_nodes(sub_G, pos_2, subcomms.keys(), node_size=500,
                       cmap=cmap, node_color=list(subcomms.values()))
labels = nx.draw_networkx_labels(sub_G,pos_2)
#nx.draw_networkx_edges(G, pos, alpha=0.5)
plt.show()
```



2. a. Calcular los motifs de hasta 5 nodos de la subred definida en el punto 1.c.

Comenezamos importando recursos necesarios para calcular lo pedido

[ ]:
```
import sys
sys.path.insert(0,'/content/drive/My Drive/social-networks-utils')

from modelos import *
from metricas import *
from homofilia import *
```

[ ]:
```
sys.path.insert(0,'/content/drive/My Drive/social-networks-utils')
from motifs.calculos import *
```

Utilizamos la siguiente función para calcular los motifs de la subred (sub_G)

[ ]:
```
calcular_motifs(sub_G,5)
```

[ ]:
```
array([  3925,    949,  14476,  23068,   1180,  24525,   7397,   1674,
        35714, 108701, 107732,  64950,  53814, 207513,   1983,  22990,
       161393,  25871,  53569,   1627,  14622,  25325,  68560,  54892,
         5852,  50265,   3156,  12129,   2024])
```

[ ]:
```
motifs = calcular_motifs(sub_G,5)
```

2. b. Calcular el promedio y desvío estandar de los motifs de una red de baseline. Calcular el significant profile de la red, y hacer un gráfico.

Para utilizar la función que calcula el promedio y el desvío estandar debo primero calcular una función que devuelva un modelo aleatorizado, como puede ser cualquiera, utilizo una de las funciones de calculos.py para calcular el modelo.

[ ]:
```
def conf():
  return configuration_model(distribucion_grados(sub_G))
```

[ ]:
```
promedio, std = motif_grafo_eleatorios(conf,5,5)
```

```
Iteracion 1
Iteracion 2; anterior: 2053.11 segs
Iteracion 3; anterior: 2122.49 segs
Iteracion 4; anterior: 1830.45 segs
Iteracion 5; anterior: 2101.85 segs
```

[ ]:
```
promedio
```

[ ]:
```
array([3.717800e+03, 3.220000e+02, 2.369320e+04, 1.662920e+04,
       4.416800e+03, 1.133060e+04, 2.341800e+03, 1.292000e+02,
       1.013934e+05, 1.883704e+05, 4.600860e+04, 7.126660e+04,
       3.796940e+04, 6.692280e+04, 1.659680e+04, 1.056632e+05,
       6.341140e+04, 5.002600e+03, 3.317100e+04, 1.335920e+04,
       4.620760e+04, 5.594800e+03, 8.113800e+03, 2.123060e+04,
       1.430000e+04, 5.421400e+03, 2.715800e+03, 7.300000e+02,
       1.800000e+01])
```

[ ]:
```
std
```

[ ]:
```
array([1.52398688e+02, 6.06630036e+00, 1.09338893e+03, 1.27018052e+03,
       6.18687126e+02, 5.44315570e+02, 1.24684241e+02, 1.73827501e+01,
       2.65830499e+03, 1.16854002e+04, 6.06529836e+03, 4.54562834e+03,
       1.00453225e+03, 6.86312009e+03, 1.02446384e+03, 1.38652506e+04,
       6.09479928e+03, 4.59324548e+02, 2.04319818e+03, 3.49941152e+03,
       5.93787819e+03, 4.38230259e+02, 9.73881184e+02, 2.17110678e+03,
       3.09912639e+03, 4.26042064e+02, 4.16287112e+02, 1.49917311e+02,
       1.21655251e+01])
```

Finalmente calculamos el significant profile de la red

[ ]:
```
sp = significance_profile(motifs, promedio, std)
sp
```

[ ]:
```
array([ 0.00967864,  0.37369564, -0.02895943,  0.02912764, -0.01885849,
        0.09628482,  0.14695522,  0.32032562, -0.08749473, -0.02196906,
        0.05900598, -0.00092907,  0.05998912,  0.09396983, -0.05141963,
       -0.02139149,  0.06831881,  0.16866586,  0.03790775, -0.01208502,
       -0.01917435,  0.1627997 ,  0.2424297 ,  0.05674906, -0.00982601,
        0.38153459,  0.0038117 ,  0.27407849,  0.59432946])
```

[ ]:
```
plt.bar(range(1,len(sp)+1),sp,color="pink",alpha=0.6)
```

[ ]:
```
<BarContainer object of 29 artists>
```

3. Detectar los roles en dicha red utilizando el algoritmo RolX, explicando el resultado obtenido.

Instalamos la librería necesaria para realizar lo pedido

[ ]:
```
pip install graphrole
```

Importamos el recurso necesario para realizar el inciso 3.

[ ]:
```
from graphrole import RecursiveFeatureExtractor, RoleExtractor
```

Aplicamos los pasos para detectar los roles

[ ]:
```
feature_extractor = RecursiveFeatureExtractor(G)
features = feature_extractor.extract_features()
role_extractor = RoleExtractor(n_roles=None)
```

[ ]:
```
role_extractor.extract_role_factors(features)
```

[ ]:
```
roles = role_extractor.roles
roles
```

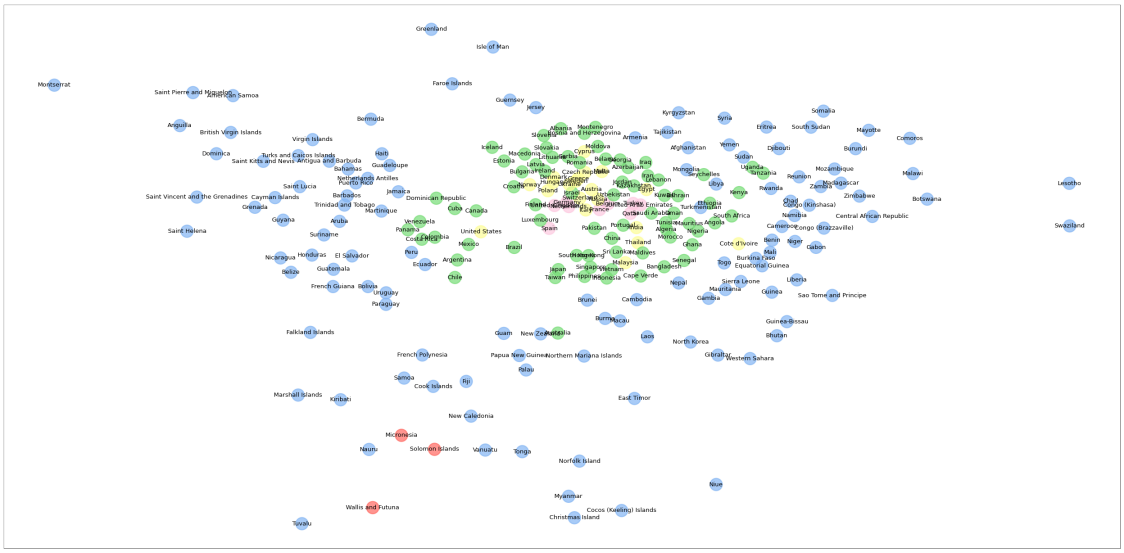
Para graficar seguimos los pasos siguientes

Creamos un diccionario que define un color para cada rol

[ ]:
```
available_colors = {'role_0': '#fdfd96', 'role_1': '#84b6f4', 'role_2':
 ↪'#77dd77', 'role_3': '#ff6961', 'role_4': '#fdcae1'}
```

[ ]:
```
colors = [available_colors[role_extractor.roles[node]] for node in G.nodes]
colors
```

[ ]:
```
pos = nx.spring_layout(G)
plt.figure(figsize = (40,20), dpi=80)
nx.draw_networkx_nodes(G,pos,nodelist=G.nodes, node_color = colors, alpha=0.7,␣
 ↪node_size=600, linewidths=2)
labels = nx.draw_networkx_labels(G,pos)
plt.show()
```



[13]:
```
#Utilizo el siguiente código para generar un pdf con la resolucion.

!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('Parcial2-TDA2.ipynb')
```

```
File ‘colab_pdf.py’ already there; not retrieving.
```

```
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab
Notebooks/Parcial2-TDA2.ipynb to pdf
[NbConvertApp] Support files will be in Parcial2-TDA2_files/
[NbConvertApp] Making directory ./Parcial2-TDA2_files
[NbConvertApp] Making directory ./Parcial2-TDA2_files
[NbConvertApp] Making directory ./Parcial2-TDA2_files
[NbConvertApp] Making directory ./Parcial2-TDA2_files
[NbConvertApp] Making directory ./Parcial2-TDA2_files
[NbConvertApp] Writing 51504 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 1066033 bytes to /content/drive/My
Drive/Parcial2-TDA2.pdf
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>
```

[13]: 'File ready to be Downloaded and Saved to Drive'