Teoría de Algoritmos II (75.30)

1.er Parcialito Domiciliario – 08/04/2022 - Fecha de Entrega: 29/04/2022

Aclaraciones: Cada ejercicio dice al final del mismo la cantidad de puntos que otorga por hacerse completamente bien (en total, 10). Se deben obtener al menos 5 puntos para aprobar, y se deben aprobar al menos 3 de los parcialitos para aprobar/regularizar la cursada. Para la fecha de entregar, enviar un mail a `mbuchwald@fi.uba.ar` con un pdf con la resolución, con nombre `P1 - PADRON.pdf`. Pueden incluir todo el material adicional que les parezca relevante (desde código hasta gráficos).

Considerando esta red que representa las conexiones de diferentes países por los vuelos (directos) realizados entre ellos, responder las siguientes preguntas. A los fines de estos ejercicios, se puede obviar la última columna del archivo csv.

1. Determinar:
   a. El diámetro de la red.
   b. El grado promedio de la red.
   c. El coeficiente de clustering promedio de la red.

   [1 punto]

2. Indicar si existe algún tipo de Homofilia y qué tipo de homofilia es. Si no hay homofilia por ningún criterio, explicar. Justificar detalladamente.

   [3 puntos]

3. Determinar los puentes (globales o locales) en dicha red.

   [1 punto]

4. a. Determinar un tipo de centralidad que podría ser útil calcular para esta red, justificando.
   b. Realizar una representación gráfica de dicha red, considerando la centralidad de los distintos países dada por la métrica del punto a (tamaño de los nodos proporcional a dicha métrica).

   [2 puntos]

5. a. Obtener una simulación de un modelado de Erdös-Rényi que corresponda a los parámetros de esta red.
   b. Obtener una simulación de un modelado de Preferential Attachment (ley de potencias) que corresponda a los parámetros de esta red.
   c. Obtener una representación de anonymous walks tanto de la red original como para las dos simuladas en los puntos a y b. Determinar por distancia coseno cuál sería la simulación más afín.

   [3 puntos]

parcialito-1

April 16, 2022

0.0.1 Parcialito 1 - Federico del Mazo - 100029

[1]:
```
import networkx as nx
import pandas as pd

df = pd.read_csv('World.csv', header=0, names=["source", "target", "_weight"])
Graphtype = nx.Graph()
G = nx.from_pandas_edgelist(df, create_using=Graphtype)
```

[2]:
```
print(f"""
1. Determinar:
  a. El diámetro de la red: {nx.diameter(G)}
  b. El grado promedio de la red: {sum([n[1] for n in G.degree()]) / len(G):.2f}
  c. El coeficiente de clustering promedio de la red: {nx.average_clustering(G):
 ↪.2f}
""")
```

```
1. Determinar:
  a. El diámetro de la red: 5
  b. El grado promedio de la red: 24.91
  c. El coeficiente de clustering promedio de la red: 0.66
```

[3]:
```
print("""
2. Indicar si existe algún tipo de Homofilia y qué tipo de homofilia es. Si no␣
 ↪hay homofilia por ningún criterio, explicar. Justificar detalladamente.
""")
```

```
2. Indicar si existe algún tipo de Homofilia y qué tipo de homofilia es. Si no
hay homofilia por ningún criterio, explicar. Justificar detalladamente.
```

[4]:
```
# Consigamos atributos de paises -> https://www.kaggle.com/datasets/
 ↪sudalairajkumar/undata-country-profiles
df2 = pd.read_csv('country_profile_variables.csv', header=0)
```

```
# Quedemonos con solo los atributos que voy a analizar, para no agregar ruido␣
 ↪al df
df2 = df2[['country', 'Region', 'Population in thousands (2017)', 'GDP per␣
 ↪capita (current US$)']]

# Agrego a manopla el atributo continente, que es al que más fé le tengo para␣
 ↪la homofilia
region_to_continent = {'SouthernAsia': 'Asia', 'SouthernEurope': 'Europe',␣
 ↪'NorthernAfrica': 'Africa', 'Polynesia': 'Oceania', 'MiddleAfrica':␣
 ↪'Africa', 'Caribbean': 'CentralAmerica', 'SouthAmerica': 'SouthAmerica',␣
 ↪'WesternAsia': 'Asia', 'Oceania': 'Oceania', 'WesternEurope': 'Europe',␣
 ↪'EasternEurope': 'Europe', 'CentralAmerica': 'CentralAmerica',␣
 ↪'WesternAfrica': 'Africa', 'NorthernAmerica': 'NorthernAmerica',␣
 ↪'SouthernAfrica': 'Africa', 'South-easternAsia': 'Asia', 'EasternAfrica':␣
 ↪'Africa', 'NorthernEurope': 'Europe', 'EasternAsia': 'Asia', 'Melanesia':␣
 ↪'Oceania', 'Micronesia': 'Oceania', 'CentralAsia': 'Asia'}
df2['Continent'] = df2['Region'].map(region_to_continent)

# Bucketeo un par de atributos, así es más facil de analizar
to_bucket = ['Population in thousands (2017)', 'GDP per capita (current US$)']
for attr in to_bucket:
    df2[attr] = pd.qcut(df2[attr], q=5).astype('str')

# Lamentablemente, nuestros 2 datasets no son perfectamente compatibles.
# Hay 13 paises con un nombre en uno, y otro nombre en otro
# También hay 16 paises de los que no tenemos datos
aliases = {"China, Hong Kong SAR": "Hong Kong", "Micronesia (Federated States␣
 ↪of)": "Micronesia", "Czechia": "Czech Republic", "Democratic People's␣
 ↪Republic of Korea": "South Korea", "Russian Federation": "Russia", "The␣
 ↪former Yugoslav Republic of Macedonia": "Macedonia", "Iran (Islamic Republic␣
 ↪of)": "Iran", "Venezuela (Bolivarian Republic of)": "Venezuela", "Brunei␣
 ↪Darussalam": "Brunei", "Falkland Islands (Malvinas)": "Falkland Islands",␣
 ↪"Syrian Arab Republic": "Syria", "Wallis and Futuna Islands": "Wallis and␣
 ↪Futuna", "Republic of Korea": "North Korea"}
df2 = df2.set_index('country').rename(index = aliases)

# Convierto mi df en un diccionario de atributos, y se lo plasmo a mi grafo
attributes = df2.to_dict('index')
nx.set_node_attributes(G, attributes)

# Para un análisis de homofilia más puro, quiero que todos mis nodos tengan␣
 ↪atributos seteados
# Borro los 16 paises que me quedaron colgados sin data
to_remove = []
for n in G.nodes(data=True):
    if not n[1]: to_remove.append(n[0])
```

```
for n in to_remove: G.remove_node(n)
```

[5]:
```
# Funciones para calcular la homofilia según atributo

import numpy as np
from collections import Counter
from itertools import combinations_with_replacement

# Dado un atributo, devuelve un diccionario con todas las proporciones de␣
 ↪aristas entre atributos.
# En el ejemplo de la sección 4.1 de Networks, Crowds, and Markets, el␣
 ↪resultado sería:
#   {(Male, Male): 11/18, (Female, Male): 5/18, (Female, Female): 3/18}
def get_attr_edges_real_fraction(G, attr):
    edges = []
    for e in G.edges():
        attr1, attr2 = G.nodes[e[0]][attr], G.nodes[e[1]][attr]
        # we sort the attributes to make sure a B-A edge counts as an A-B one
        edges.append(tuple(sorted([attr1,attr2])))
    count = Counter(edges)
    total_edges = nx.number_of_edges(G)
    return {k: v/total_edges for k,v in count.items()}

# Dado un atributo, devuelve un diccionario con todas las proporciones de␣
 ↪aristas ideales si no hubiese homofilia.
# Es decir, de todos los nodos con sus distintas probabilidades de tomar algun␣
 ↪valor del atributo,
#   la arista entre dos nodos del mismo atributo tendrá p*p / cant_nodos de␣
 ↪aparecer,
#   y la arista entre dos nodos de distinto atributo tendrá 2*p*q / cant_nodos␣
 ↪de aparecer,
# En el ejemplo de la sección 4.1 de Networks, Crowds, and Markets, el␣
 ↪resultado sería:
#   {(Male, Male): 4/9, (Female, Male): 4/9, (Female, Female): 1/9}
def get_attr_edges_expected_fraction(G, attr):
    attr_count = Counter(nx.get_node_attributes(G, attr).values())
    total_nodes = nx.number_of_nodes(G)
    attr_fraction = {k: v/total_nodes for k,v in attr_count.items()}

    attr_combinations = combinations_with_replacement(attr_count.keys(), 2)
    count = {}
    for attr1, attr2 in attr_combinations:
        if (attr1 == attr2):
            expected_fraction = attr_fraction[attr1] * attr_fraction[attr2]
        else:
```

```
            expected_fraction = attr_fraction[attr1] * attr_fraction[attr2] * 2
        count[tuple(sorted([attr1, attr2]))] = expected_fraction
    return count


# Dado un atributo, devuelve un porcentaje que simboliza cuan homofílico␣
 ↪respecto del atributo es el grafo
# Es decir, en un grafo donde no hay nada de homofilia, este valor será 0%,
#   y en un grafo donde hay toda la homofilia del mundo, este valor será 100%
# (ojo, este valor es exactamente el inverso al que aprendimos en clase!)
# Al lidiar con atributos multivariados, el resultado final será un promedio␣
 ↪ponderado de todos los
#   coeficientes de valor_real/valor_esperado para cada arista que junta un par␣
 ↪de atributos.
# La ponderación de cada arista es la cantidad total de nodos que pertenecen a␣
 ↪los atributos que une.
# En el ejemplo de la sección 4.1 de Networks, Crowds, and Markets, el␣
 ↪resultado sería:
#   1 - 0.62 ==> 38%
def homophily_percentage(G, attr):
    attributes = Counter(nx.get_node_attributes(G, attr).values())
    expected = get_attr_edges_expected_fraction(G, attr)
    real = get_attr_edges_real_fraction(G, attr)

    percentages = []
    weights = []
    for attr1, attr2 in expected:
        # En el estudio de homofilia, salteo las aristas entre el mismo valor
        if attr1 == attr2: continue

        k = tuple(sorted([attr1, attr2]))
        percentage = real.get(k, 0) / expected[k]
        weight = attributes[attr1] + attributes[attr2]
        percentages.append(percentage)
        weights.append(weight)
    return (1 - np.average(percentages, weights=weights)) * 100
```

[6]:
```
attrs = ['Continent', 'Region', 'Population in thousands (2017)', 'GDP per␣
 ↪capita (current US$)']

for attr in attrs:
    print(f"El grafo tiene un {homophily_percentage(G, attr):.2f}% de homofilia␣
 ↪por la característica {attr}")
```

```
# Donde esperabamos encontrar un gran porcentaje de homofilia era en en la␣
 ↪homofilia por continentes, que se cumple. Después de eso, creí que al menos␣
 ↪habría una homofilia más alta según población (o según PBI per capita, con␣
 ↪la idea de que paises más ricos suelen tener mas viajes entre sí).
```

```
El grafo tiene un 47.64% de homofilia por la característica Continent
El grafo tiene un 14.66% de homofilia por la característica Region
El grafo tiene un 12.48% de homofilia por la característica Population in
thousands (2017)
El grafo tiene un 8.61% de homofilia por la característica GDP per capita
(current US$)
```

[7]:
```
# Ya dejamos de jugar con los atributos y la homofilia, recuperemos el grafo␣
 ↪original!
G = nx.from_pandas_edgelist(df, create_using=Graphtype)
```

[8]:
```
print(f"""
3. Determinar:
  a. Puentes globales: {list(nx.bridges(G))}
  b. Puentes locales: {[b for b in list(nx.local_bridges(G)) if b[2] !=␣
 ↪float('inf')]}
""")
```

```
3. Determinar:
  a. Puentes globales: [('Fiji', 'Tuvalu'), ('United States', 'American Samoa'),
('United Kingdom', 'Saint Helena'), ('Canada', 'Saint Pierre and Miquelon'),
('Antigua and Barbuda', 'Montserrat'), ('New Zealand', 'Niue'), ('South Africa',
'Lesotho'), ('South Africa', 'Swaziland'), ('Burma', 'Myanmar')]
  b. Puentes locales: [('Papua New Guinea', 'Micronesia', 3), ('Micronesia',
'Marshall Islands', 3)]
```

[9]:
```
print("""
4.
  a. Determinar un tipo de centralidad que podría ser útil calcular para esta␣
 ↪red, justificando.
  b. Realizar una representación gráfica de dicha red, considerando la␣
 ↪centralidad de los distintos países dada por la métrica del punto a (tamaño␣
 ↪de los nodos proporcional a dicha métrica).
""")
```

```
4.
  a. Determinar un tipo de centralidad que podría ser útil calcular para esta
red, justificando.
  b. Realizar una representación gráfica de dicha red, considerando la
```

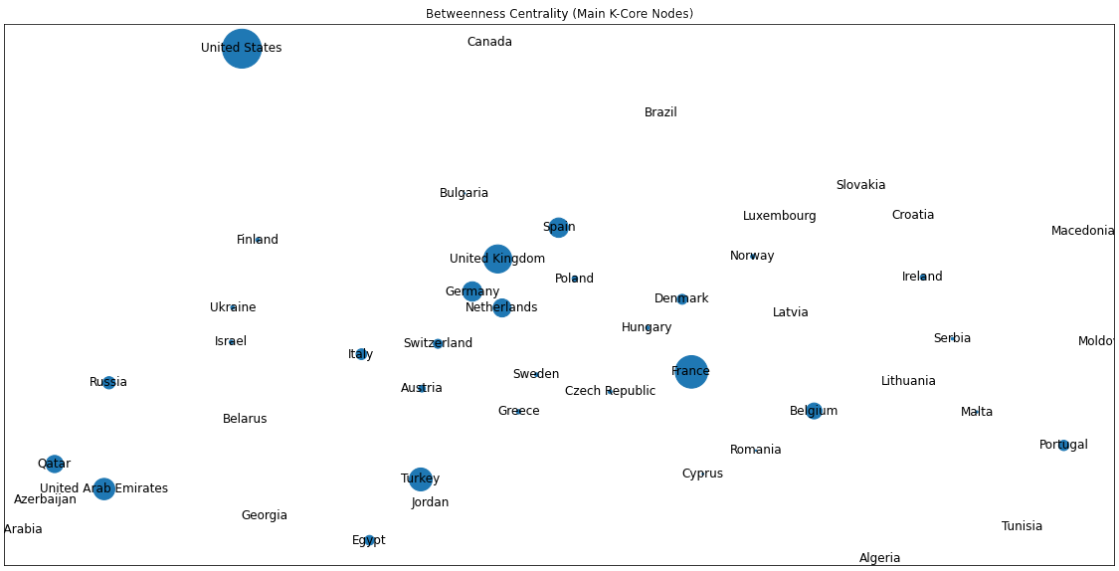
centralidad de los distintos países dada por la métrica del punto a (tamaño de los nodos proporcional a dicha métrica).

[10]:
```
# Un análisis que quiero hacer es 'si un día cierro X aeropuerto, cuantos viajes estoy disrumpiendo?',
#   esto lo puedo ver con Betweenness: cuáles son los nodos por los que más paso si voy de X a Y?
#   y por ende cerrar estos aeroupertos va a hacer que haya que modificar los caminos de X a Y
#   (o sea, los aeropuertos adyacentes van a tener que buscar otro lugar en común para hacer escala)
#   (esta idea no es tan lejana a buscar puentes locales...)

# Voy a graficar solo los nodos que están en el k-core principal, para que el gráfico se pueda ver bien
#  (si no, es un choclo de 230 nodos que no se entiende para nada)
import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))

last_core = nx.k_core(G).nodes()
centrality = nx.betweenness_centrality(G)
nodes = {k:v*10000 for k,v in centrality.items() if k in last_core}

plt.title("Betweenness Centrality (Main K-Core Nodes)")
nx.draw_networkx(G, nodelist=nodes.keys(), node_size=list(nodes.values()), edgelist=[])
```



[11]:
```
print("""
5.
  a. Obtener una simulación de un modelado de Erdös-Rényi que corresponda a los␣
 ↪parámetros de esta red.
  b. Obtener una simulación de un modelado de Preferential Attachment (ley de␣
 ↪potencias) que corresponda a los parámetros de esta red.
  c. Obtener una representación de anonymous walks tanto de la red original␣
 ↪como para las dos simuladas en los puntos a y b. Determinar por distancia␣
 ↪coseno cuál sería la simulación más afín.
""")
```

```
5.
  a. Obtener una simulación de un modelado de Erdös-Rényi que corresponda a los
parámetros de esta red.
  b. Obtener una simulación de un modelado de Preferential Attachment (ley de
potencias) que corresponda a los parámetros de esta red.
  c. Obtener una representación de anonymous walks tanto de la red original como
para las dos simuladas en los puntos a y b. Determinar por distancia coseno cuál
sería la simulación más afín.
```

[12]:
```
import math

def nCr(n,r):
    f = math.factorial
    return f(n) // f(r) // f(n-r)

n_nodes = G.number_of_nodes()
n_edges = G.number_of_edges()
total_possible_edges = nCr(n_nodes, 2)
avg_degree = sum([n[1] for n in G.degree()]) / len(G)

erdos = nx.erdos_renyi_graph(n_nodes, n_edges / total_possible_edges)
barabara = nx.barabasi_albert_graph(n_nodes, n_edges // n_nodes)

grafos = {
    "Original Graph": G,
    "Erdös-Rényi": erdos,
    "Barabási-Albert": barabara
}

for k,v in grafos.items():
    print(f"{k}: {v}")
```

```
Original Graph: Graph with 229 nodes and 2852 edges
Erdös-Rényi: Graph with 229 nodes and 2898 edges
```

```
Barabási-Albert: Graph with 229 nodes and 2604 edges
```

[13]:
```python
# https://github.com/nd7141/AWE
from AnonymousWalkKernel import AnonymousWalks
from scipy import spatial

length = 5
embeds = {}
for name, g in grafos.items():
    emb, meta = AnonymousWalks(g).embed(steps = length, method = 'sampling',␣
 ↪keep_last=True, verbose=False)
    embeds[name] = emb

simils = {}
for name in ["Erdös-Rényi","Barabási-Albert"]:
    simils[name] = 1 - spatial.distance.cosine(embeds[name], embeds["Original␣
 ↪Graph"])

for name, simil in simils.items():
    print(f"Similitud Coseno entre {name} y nuestro OG: {simil}")

print(f"Ganador: {max(simils, key=simils.get)}!")
```

```
Similitud Coseno entre Erdös-Rényi y nuestro OG: 0.9995770735792
Similitud Coseno entre Barabási-Albert y nuestro OG: 0.9997761192376725
Ganador: Barabási-Albert!
```