Trabajo Práctico 1 - Teoría de Algoritmos II

Nombre: Mateo Capón Blanquer

Padrón: 104258

Enunciado

Aclaraciones: Cada ejercicio dice al final del mismo la cantidad de puntos que otorga por hacerse completamente bien (en total, 15). Se deben obtener al menos 9 puntos para aprobar. Para la fecha de entregar, enviar un mail a mbuchwald@fi.uba.ar con un pdf con la resolución, con nombre TP1-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.

Análisis Introductorio y Preprocesamiento de los datos

In [1]:
```
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt
import geopandas as gpd
import sys
sys.path.append('./social-networks-utils')


df = pd.read_csv('World.csv')

# Leo este dataset de paises con más información que la dada en el dataset Worl
# Fuente: https://www.naturalearthdata.com/downloads/110m-cultural-vectors/110m
gdf = gpd.read_file("ne_110m_admin_0_countries")

# Uso centroid solo para graficar -> Ignoro warnings.
gdf["x"] = gdf['geometry'].centroid.x
gdf["y"] = gdf['geometry'].centroid.y
```

```
/tmp/ipykernel_12246/584385171.py:15: UserWarning: Geometry is in a geographic
CRS. Results from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to
re-project geometries to a projected CRS before this operation.

  gdf["x"] = gdf['geometry'].centroid.x
/tmp/ipykernel_12246/584385171.py:16: UserWarning: Geometry is in a geographic
CRS. Results from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to
re-project geometries to a projected CRS before this operation.

  gdf["y"] = gdf['geometry'].centroid.y
```

Importante: En el Dataset hay una arista de "Burma" - "Myanmar", la cual me llamó mucho la atención, porque entiendo que son dos formas de referirse al mismo país. Decidí interpretar esta arista como un vuelo interno (un loop en Myanmar).

Esto hizo que, por ejemplo, el diámetro de la red baje en un una unidad.

In [2]:
```
# Limpio algunos datos de World.csv para poder hacer merge con dataset con mas
to_replace = {
    "United States": "United States of America",
    "China": "People's Republic of China",
    "Cote d'Ivoire": "Ivory Coast",
    "Bahamas": "The Bahamas",
    "Gambia": "The Gambia",
    "Macedonia": "North Macedonia",
    "Congo (Kinshasa)": "Democratic Republic of the Congo",
    "Congo (Brazzaville)": "Republic of the Congo",
    "Sao Tome and Principe": "São Tomé and Príncipe",
    "Burma": "Myanmar",
    "Netherlands Antilles": "Curaçao",
}
df = df.replace(to_replace)

# Uso nx.Graph porque son vuelos entre ellos (no dirigido).
G = nx.from_pandas_edgelist(df, edge_attr='ConexionAeropuertos',\
                            source='Origen', target='Destino', create_using=nx.
```

In [3]:
```
print("Los nodos que no se encuentran en el dataframe con mas info son: ")
no_origin = set(df[~ df["Origen"].isin(gdf["NAME_EN"])]["Origen"])
no_dest = set(df[~ df["Destino"].isin(gdf["NAME_EN"])]["Destino"])
no_origin.union(no_dest)
```

Out[3]:
```
Los nodos que no se encuentran en el dataframe con mas info son:
{'Christmas Island',
 'Cocos (Keeling) Islands',
 'French Guiana',
 'Guadeloupe',
 'Martinique',
 'Mayotte',
 'Micronesia',
 'Reunion',
 'Swaziland',
 'Virgin Islands'}
```

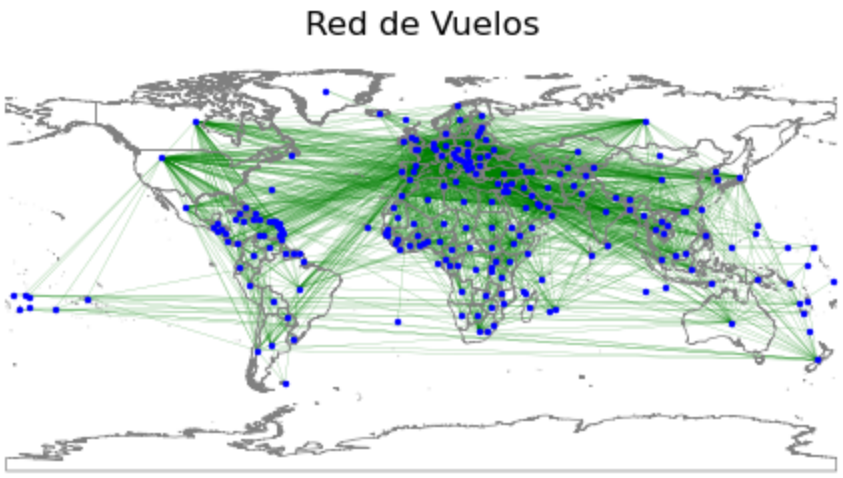
In [4]:
```
# Agrego información para esos nodos, haciendo busquedas en google maps.
# Las coordenadas que agrego son de puntos al azar dentro del pais / isla (solo

rows_to_add = [
    {"NAME_EN": "Martinique", "y": 14.636383, "x":-60.985519, "CONTINENT": "Nor
    {"NAME_EN": "Reunion", "y": -21.133153, "x": 55.525737, "CONTINENT": "Afric
    {"NAME_EN": "French Guiana", "y":4.299076, "x":-52.972789, "CONTINENT": "So
    {"NAME_EN": "Guadeloupe", "y":16.233781, "x": -61.633313, "CONTINENT": "Nor
    {"NAME_EN": "Christmas Island", "y":-10.481556, "x": 105.652649, "CONTINENT
    {"NAME_EN": "Micronesia", "y":6.881294, "x": 158.219760, "CONTINENT": "Ocea
    {"NAME_EN": "Mayotte", "y":-12.805829, "x": 45.161262, "CONTINENT": "Africa
    {"NAME_EN": "Virgin Islands", "y":18.419666, "x":-64.633546, "CONTINENT": "
    {"NAME_EN": "Swaziland", "y":-26.458953, "x": 31.476765, "CONTINENT": "Afri
    {"NAME_EN": "Cocos (Keeling) Islands", "y":-12.152164, "x":96.869617, "CONT
]
```

```
df_more_data = gdf[["NAME_EN", "x", "y", "CONTINENT"]]
df_more_data = pd.concat([df_more_data, pd.DataFrame(rows_to_add)])
df_more_data.head(5)
```

Out[4]:

	NAME_EN	x	y	CONTINENT
0	Indonesia	117.270433	-2.222961	Asia
1	Malaysia	109.698868	3.792393	Asia
2	Chile	-71.364375	-37.743607	South America
3	Bolivia	-64.684754	-16.706877	South America
4	Peru	-74.378065	-9.154388	South America

In [5]:
```
world = gdf.boundary.plot(color='gray', linewidth=0.5)
pos = {row["NAME_EN"]: (row["x"], row["y"]) for _,row in df_more_data.iterrows(
nx.draw(G, pos, with_labels=False, width= 0.1, node_size=2, node_color='blue',

plt.title('Red de Vuelos')
plt.show()
```



In [40]:
```
print(f"La red tiene {G.number_of_nodes()} paises asumiendo Burma == Myanmar")
```

```
La red tiene 228 paises asumiendo Burma == Myanmar
```

Ejercicio 1

Determinar:

a. El diámetro de la red.

In [7]:
```
diameter = nx.diameter(G)
print(f"El diámetro de la red es {diameter}")
```

```
El diámetro de la red es 4
```

Con tan solo 4 vuelos se puede llegar a cualquier país / isla de la red. Si se incluyese el nodo Burma - Myanmar, daría 5.

b. El grado promedio de la red.

In [8]:
```
mean_degree = sum(dict(G.degree()).values()) / len(G)
print(f"El grado promedio de la red es {mean_degree}")
```

```
El grado promedio de la red es 25.017543859649123
```

c. El coeficiente de clustering promedio de la red.

In [9]:
```
clustering_coef = nx.average_clustering(G)
print(f"El coeficiente de clustering promedio de la red es {clustering_coef}")
```

```
El coeficiente de clustering promedio de la red es 0.6635981973803952
```

Este coeficiente de clustering es bastante alto. El 66% de los triangulos que se podrían formar, se están formando.

Ejercicio 2

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

Hipótesis: Hay homofilia por Continente

Para determinar si mi hipótesis es cierta, calculo:

p_i = Probabilidad de que un nodo sea del continente i.

P = Probabilidad de que dos nodos estén en el mismo continente

In [10]:
```
df_nodes_merged = pd.merge(pd.DataFrame({"NAME_EN":list(G.nodes())}), df_more_c
df_nodes_merged["CONTINENT"].value_counts()
```

Out[10]:
```
Africa                        55
Asia                          49
Europe                        44
North America                 37
Oceania                       25
South America                 14
Seven seas (open ocean)        4
Name: CONTINENT, dtype: int64
```

Dado que los paises / islas de "Seven seas (open ocean)" no se encuentran necesariamente en la misma zona, y son pocos, los quito del grafo para que no generen ruido en la medición. Es muy probable que, al ser tan pocos, todos los viajes de estos paises / islas, sean a otro continente.

In [11]:
```
nodes_to_delete = list(df_nodes_merged[df_nodes_merged["CONTINENT"] == "Seven s
print("Los nodos a eliminar son: ")
nodes_to_delete
```

```
Los nodos a eliminar son:
```

Out[11]:
```
['Maldives', 'Mauritius', 'Seychelles', 'Saint Helena']
```

In [12]:
```
G_homofilia_analysis = G.copy()
G_homofilia_analysis.remove_nodes_from(nodes_to_delete)
df_nodes_merged = df_nodes_merged[df_nodes_merged["CONTINENT"] != "Seven seas (
```

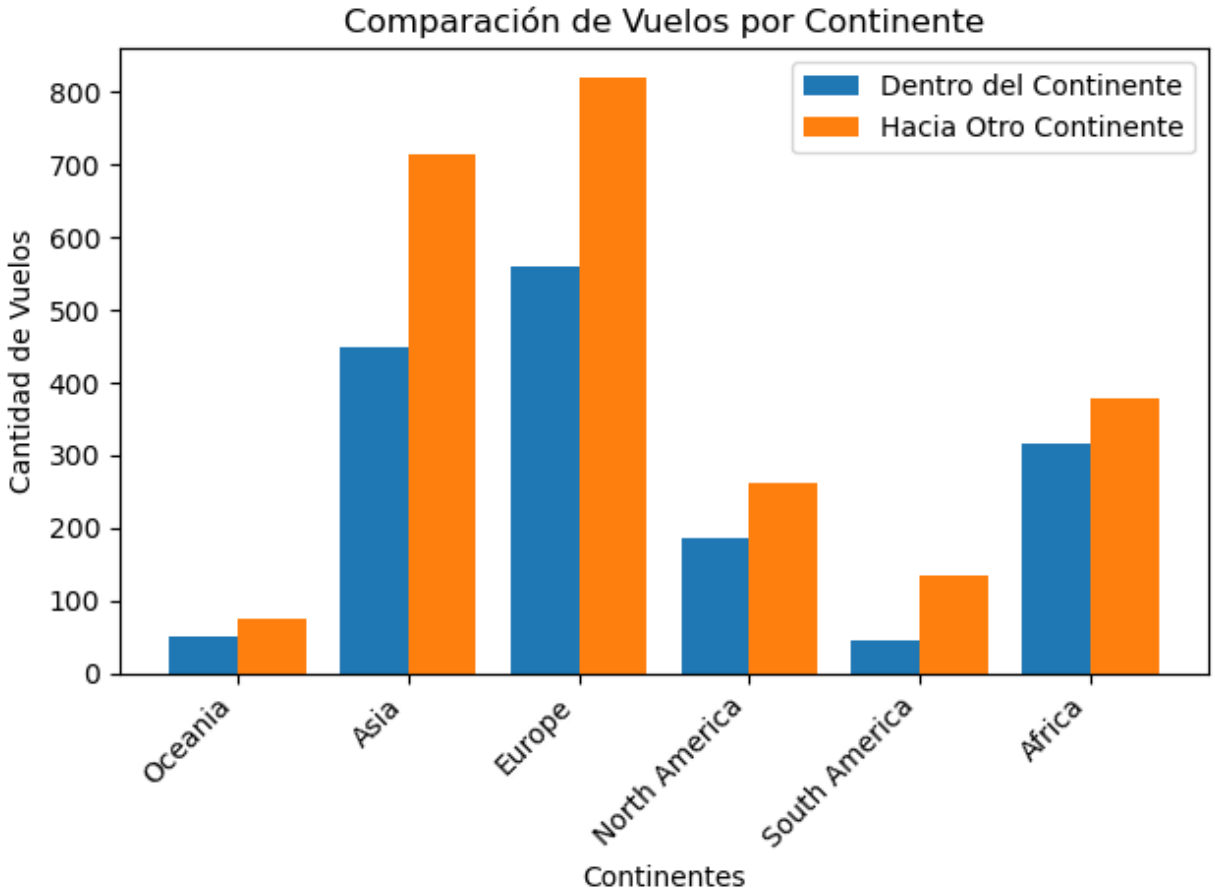
In [13]:
```
p_i = dict(df_nodes_merged["CONTINENT"].value_counts() / G.number_of_nodes())
print(f"La probabilidad estimada de p_i es: ")
display(p_i)

P = sum([p**2 for p in p_i.values()])
threshold = 1 - P
print(f"La probabilidad P estimada de que dos nodos estén en el mismo continent
print(f"El threshold de no-homofilia estimado 1-P es: {threshold}")
```

```
La probabilidad estimada de p_i es:
{'Africa': 0.2412280701754386,
 'Asia': 0.2149122807017544,
 'Europe': 0.19298245614035087,
 'North America': 0.16228070175438597,
 'Oceania': 0.10964912280701754,
 'South America': 0.06140350877192982}
La probabilidad P estimada de que dos nodos estén en el mismo continente es:
0.18374884579870732
El threshold de no-homofilia estimado 1-P es: 0.8162511542012927
```

In [14]:
```
from collections import defaultdict

node_attributes = df_nodes_merged.set_index("NAME_EN").to_dict("index")
nx.set_node_attributes(G_homofilia_analysis, node_attributes)

same_continent_edges = defaultdict(int)
different_continent_edges = defaultdict(int)
for u, v in G_homofilia_analysis.edges():
    u_cont = G_homofilia_analysis.nodes[u]["CONTINENT"]
    v_cont = G_homofilia_analysis.nodes[v]["CONTINENT"]
    if u_cont == v_cont:
        same_continent_edges[u_cont] += 1
    else:
        # cuento dos veces la misma arista. Una en el origen y otra en el dest
        different_continent_edges[u_cont] += 1
        different_continent_edges[v_cont] += 1
```

In [15]:
```
import matplotlib.pyplot as plt

conts = list(same_continent_edges.keys())

hight_same = [same_continent_edges[cont] for cont in conts]
hight_diff = [different_continent_edges[cont] for cont in conts]

pos = range(len(conts))
width = 0.4

plt.bar([p - width / 2 for p in pos], hight_same, width, label='Dentro del Cont
plt.bar([p + width / 2 for p in pos], hight_diff, width, label='Hacia Otro Cont

plt.xlabel('Continentes')
plt.ylabel('Cantidad de Vuelos')
```

```
plt.xticks(pos, conts, rotation=45, ha='right')
plt.legend()
plt.title('Comparación de Vuelos por Continente')
plt.tight_layout()
plt.show()
```



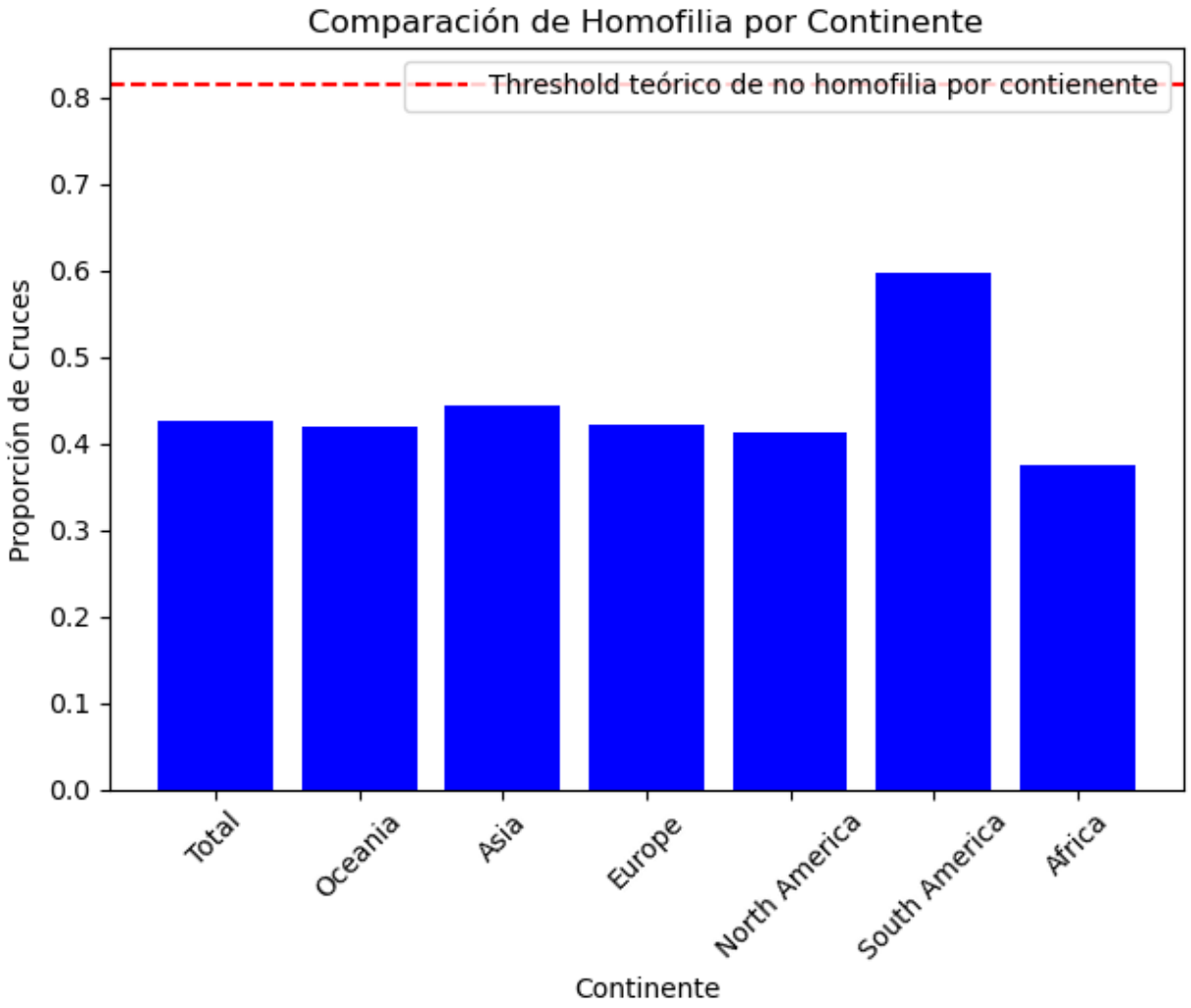
Como se puede observar en el gráfico, mi hipótesis pareciera cumplirse. Aunque no suele haber más vuelos intra-continentes que inter-continentes, estos valores se asemejan mucho, y evidentemente hay homofilia. Si no hubiese homofilia, la altura de las barras dentro del continente debería ser, en general, mucho menor, proporcionales a la cantidad de nodos del respectivo continente.

Estas afirmaciones se pueden observar también en el próximo gráfico, donde se muestra el threshold previamente calculado.

In [16]:
```
props_cont = {}
prop_total = (G_homofilia_analysis.number_of_edges() - sum(same_continent_edges
props_cont["Total"] = prop_total

for cont in conts:
    # Divido por dos para darle el mismo peso a todas las aristas
    props_cont[cont] = different_continent_edges[cont] / (2 * same_continent_ed

props_cont
```

Out[16]:
```
{'Total': 0.4259656652360515,
 'Oceania': 0.42045454545454547,
 'Asia': 0.44382371198013654,
 'Europe': 0.4223826714801444,
 'North America': 0.41232227488151657,
 'South America': 0.5982142857142857,
 'Africa': 0.37561942517343905}
```

In [17]:
```
bars = list(props_cont.keys())
plt.bar(bars, props_cont.values(), color='b')
plt.xlabel('Continente')
plt.ylabel('Proporción de Cruces')
plt.title('Comparación de Homofilia por Continente')

plt.axhline(y=threshold, color='r', linestyle='--', label='Threshold teórico de
plt.legend()
plt.tight_layout()
plt.xticks(rotation=45)
plt.show()
```



En esta visualización se puede ver muy claramente que sí hay homofilia por continente. Mi hipótesis se valida en la red.

Sudamérica es compensada un poco por los otros continentes para el valor total obtenido. Sin embargo, los valores para todos los continentes son similares, con un pico mucho menor (casi

un medio) respecto al threshold teórico de no-homofilia.

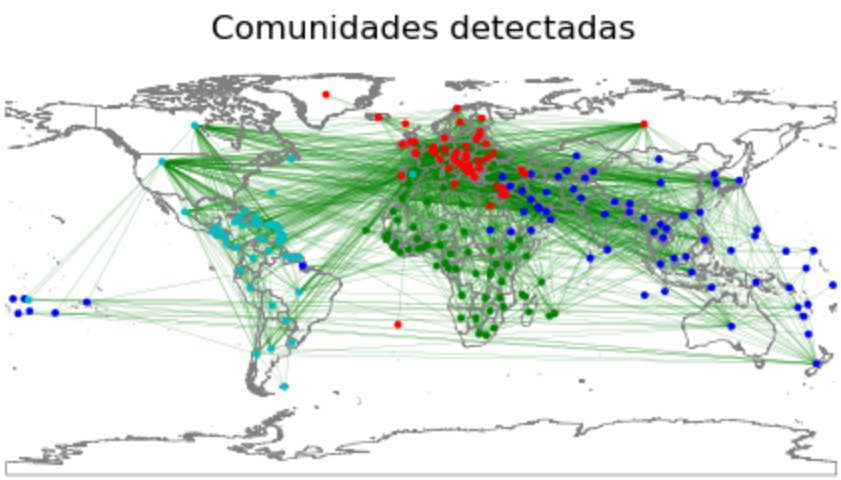
b. Obtener una visualización de las comunidades presentes en dicha red (indicando el algoritmo utilizado).

Utilizo el algoritmo de Louvain que considera la métrica de modularidad para obtener las comunidades.

In [18]:
```
communities = nx.community.louvain_communities(G, seed=42)
```

In [19]:
```
node_community_mapping = {}
for i, community in enumerate(communities):
    for node in community:
        node_community_mapping[node] = i

colors = ['r', 'g', 'b', 'c', 'm', 'y', 'k']

node_colors = [colors[node_community_mapping[node]] for node in G.nodes]
```

In [20]:
```
world = gdf.boundary.plot(color='gray', linewidth=0.5)
pos = {row["NAME_EN"]: (row["x"], row["y"]) for _,row in df_more_data.iterrows(
nx.draw(G, pos, with_labels=False,node_color=node_colors, width= 0.1, node_size

plt.title('Comunidades detectadas')
plt.show()
```



En el gráfico de communities se ve muy claramente cómo se detectaron 4 comunidades:

1 - América (Celeste)

2 - Europa (Rojo)

3 - África (Verde)

4 - Asia y Oceanía (Azul)

Hay muy pocos "outliers". Se puede observar a una Guayana de color azul en América, y un pequeño punto celeste en Europa. Fuera de estos dos puntos, la comunidades están muy bien

marcadas por los continentes.

c. Considerando lo mencionado en el punto (2.a), ¿corresponde lo encontrado por el algoritmo de detección de comunidades en relación a lo indicado en dicho punto? ¿Se verifica que efectivamente hay homofilia, o no, según hayas indicado? ¿como contrastan este resultado con lo indicado (y calculado) anteriormente?

Los resultados obtenidos en el punto (b) corroboran aún más el hecho de que hay homofilia por continente. Los resultados son contundentes. El algoritmo de detección de comunidades detectó 4 comunidades. Por su parte, yo separé a America del Norte de América del Sur, y a Asia de Oceanía.

En el punto (a) se demostró utilizando el threshold teórico que hay homofilia por continente. En el punto (b) se muestra esto gráficamente utilizando el algoritmo de Louvain.

A pesar de que son resultados esperados por la naturaleza del set de datos (cercanía entre países del mismo continente), me sorprende mucho la separación que hizo el algoritmo de detección de comunidades.

Ejercicio 3

a. Determinar un tipo de centralidad que podría ser útil calcular para esta red, justificando.

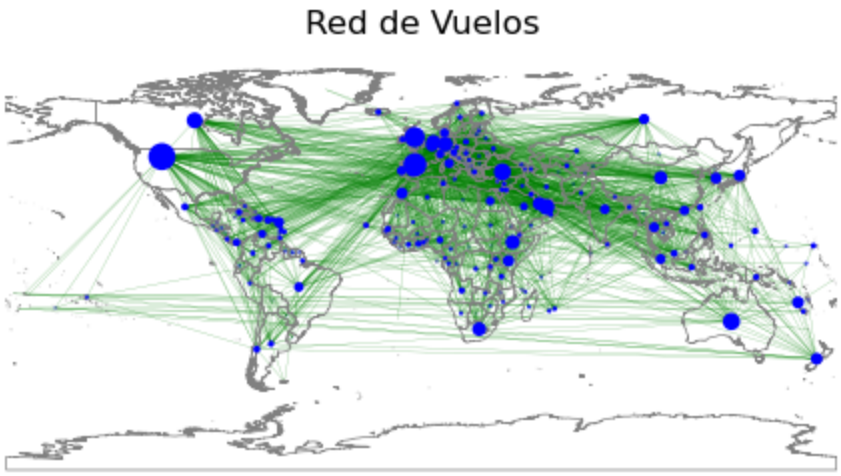
Un tipo de centralidad que me parece útil calcular es la de Betweenness Centrality, ya que se trata de un mapa que conecta países. A través de esta métrica se pueden observar aquellos paises que son centrales para la conexión del mundo.

Otro punto interesante a observar de esta métrica es si hay muchos países con una centralidad alta similar, o si hay pocos países con alta centralidad. Este último caso podría implicar un problema, ya que un conflicto en los aeropuertos de ese país (o determinada política) podría generar una rápida desconexión del mundo a través de la vía area. Además, si un solo país es centro de la conexión, implicaría que tiene demasiado poder.

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).

In [21]:
```
betweenness_centrality = nx.betweenness_centrality(G)
```

In [22]:
```
max_node_size = 500

node_size = [max_node_size * betweenness_centrality[node] for node in G.nodes]

world = gdf.boundary.plot(color='gray', linewidth=0.5)
pos = {row["NAME_EN"]: (row["x"], row["y"]) for _,row in df_more_data.iterrows(
nx.draw(G, pos, with_labels=False, width= 0.1, node_size=node_size, node_color=

plt.title('Red de Vuelos')
plt.show()
```

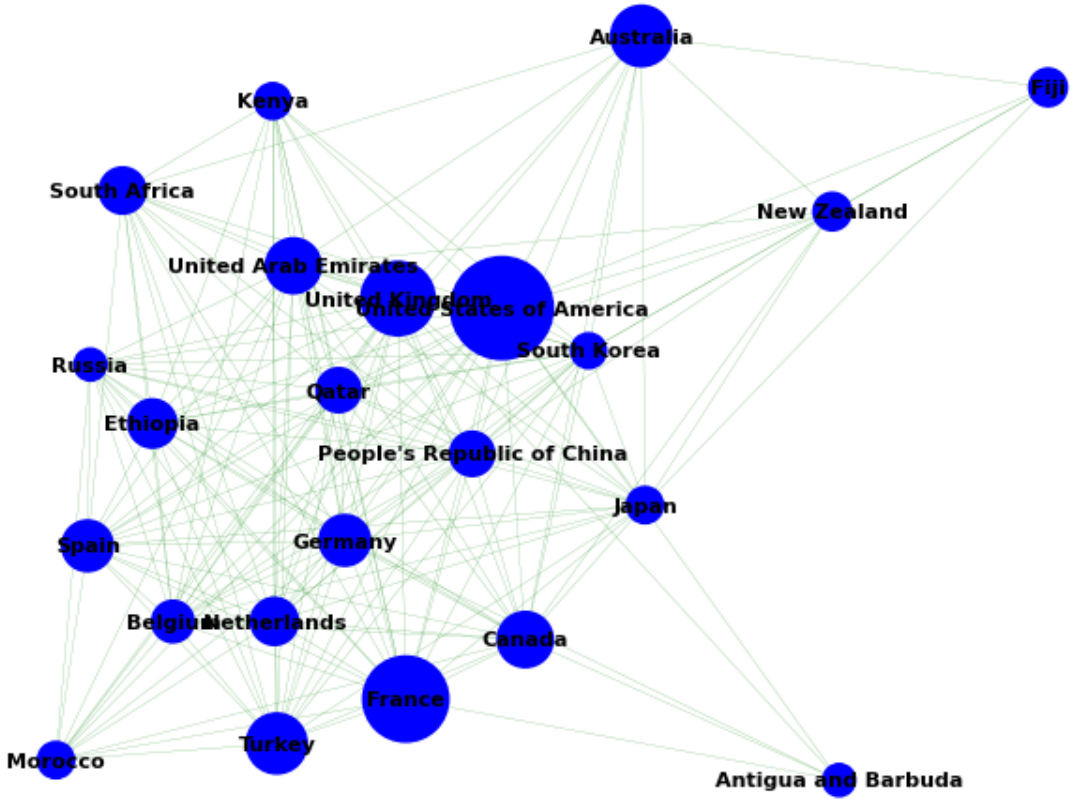


En esta visualización se puede observar que muchos de los países con mayor centralidad se encuentran en Europa. Esto era de esperarse ya que uno podría pensar a Europa como un "punto medio" en el globo: el océano pacífico es más ancho (cantidad de meridianos) que el atlántico. Está idea sin embargo no se cumple con los países de África, lo cual es también interesante. Esto hace pensar que, en general, los habitantes de países Africanos, dependen muchas veces de hacer una escala en Europa para recorrer el mundo.

Otro punto que me parece interesante remarcar es que en Oceanía y América del Norte (incluyendo Caribe y América Central) hay claramento dos países que se destacan: Estados Unidos y Australia. Estos parecen ser nexos para la conexión del mundo hacia sus continentes.

A continuación hago una visualización con únicamente los países que tuvieron una mayor betweenness centrality, para poder visualizarlos más claramente.

In [23]:
```
max_node_size = 10000
G_betweenness = G.copy()
node_size = []
for k, v in betweenness_centrality.items():
    if v < 0.015:
        G_betweenness.remove_node(k)
    else:
        node_size.append(max_node_size * v)

nx.draw(G_betweenness, width= 0.1, with_labels=True, node_size=node_size, node_
```



De este último gráfico me parece interesante el hecho de que no hay ningún país de Sudamérica.

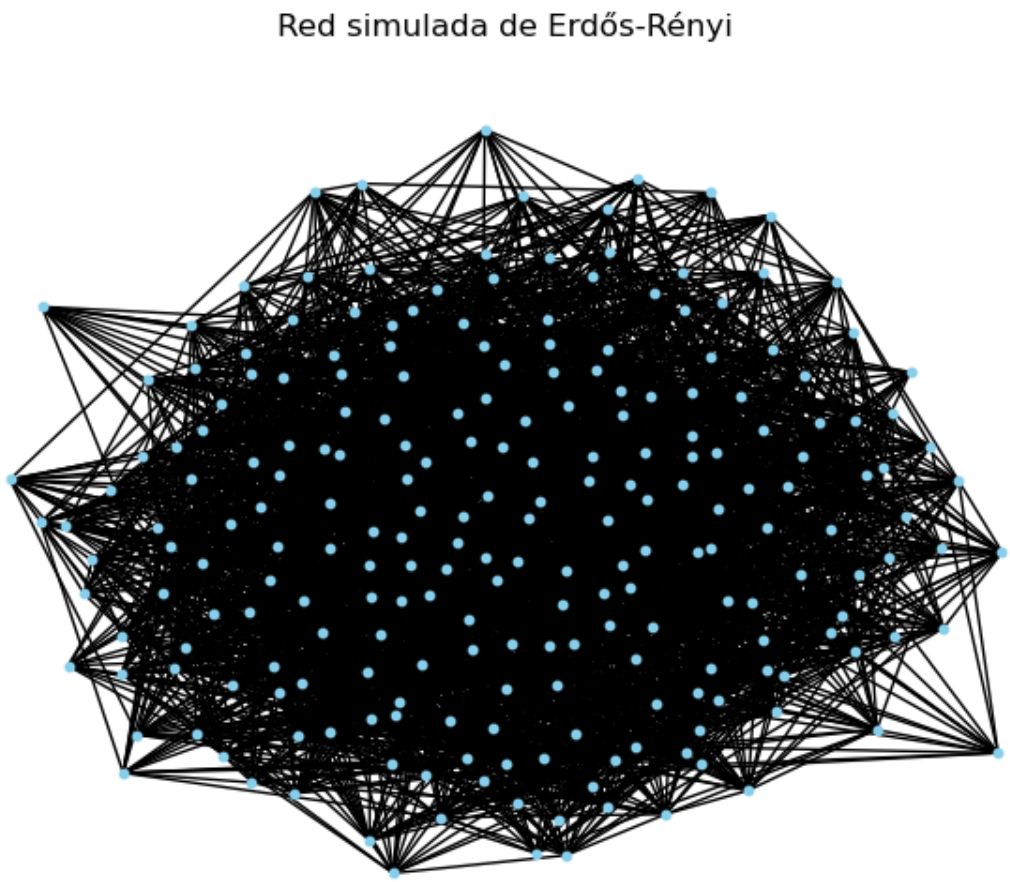
Ejercicio 4

a. Obtener una simulación de un modelado de Erdös-Rényi que corresponda a los parámetros de esta red.

In [24]:
```
n = G.number_of_nodes()
p = mean_degree / (n - 1)

# Simulación Erdös Rényi con n nodos y con proabilidad p de que la arista u,v e
G_erdos = nx.erdos_renyi_graph(n, p)
```

In [25]:
```
pos = nx.spring_layout(G_erdos)
nx.draw(G_erdos, pos, node_size=10, node_color='skyblue')
plt.title("Red simulada de Erdős-Rényi")
plt.show()


mean_degree_erdos = sum(dict(G_erdos.degree()).values()) / len(G_erdos)
clust_erdos = nx.average_clustering(G_erdos)
diameter_erdos = nx.diameter(G_erdos)
shortest_path_dist = nx.average_shortest_path_length(G)
shortest_path_erdos = nx.average_shortest_path_length(G_erdos)
```

```
connected_components = len(list(nx.connected_components(G)))
connected_components_erdos = len(list(nx.connected_components(G_erdos)))
```



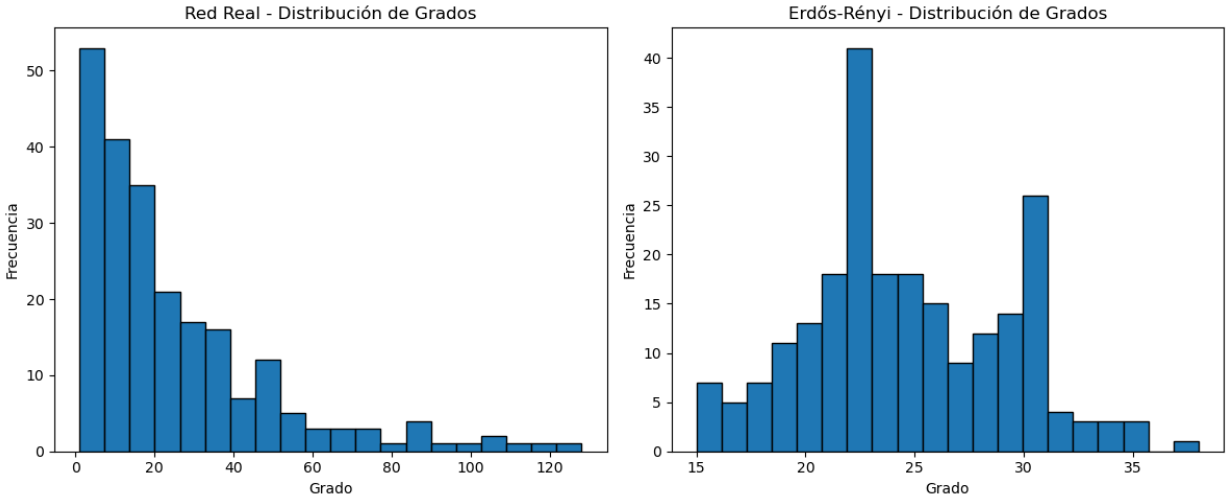
In [26]:
```
print("Resultados".center(70, "-"))
print("Calculo".center(20) + "Red Real".center(20) + "Erdős-Rényi".center(20))
print("Coef. Clustering".ljust(20) + "{:.2f}".format(clustering_coef).rjust(20)
print("Diametros".ljust(20) + f"{diameter}".rjust(20) + f"{diameter_erdos}".rju
print("Distancia Promedio".ljust(20) + "{:.2f}".format(shortest_path_dist).rjus
print("Grado Promedio".ljust(20) + "{:.2f}".format(mean_degree).rjust(20) + "{
print("Componentes Conexas".ljust(20) + f"{connected_components}".rjust(20) + 
```

```
------------------------------Resultados------------------------------
      Calculo             Red Real          Erdős-Rényi
Coef. Clustering                0.66                 0.11
Diametros                          4                    3
Distancia Promedio              2.24                 1.95
Grado Promedio                 25.02                24.59
Componentes Conexas                1                    1
```

Tal como fue observado en clase, Erdős-Rényi se comporta similar a la red real con respecto a Distancias y Grados, pero el coeficiente de clustering, y la distribución de los grados es muy diferente.

In [27]:
```
degrees = [degree for node, degree in G.degree()]
degrees_erdos = [degree for node, degree in G_erdos.degree()]

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
```

```
plt.hist(degrees, bins=20, edgecolor='k')
plt.title("Red Real - Distribución de Grados")
plt.xlabel("Grado")
plt.ylabel("Frecuencia")

plt.subplot(1, 2, 2)
plt.hist(degrees_erdos, bins=20, edgecolor='k')
plt.title("Erdős-Rényi - Distribución de Grados")
plt.xlabel("Grado")
plt.ylabel("Frecuencia")

plt.tight_layout()
plt.show()
```

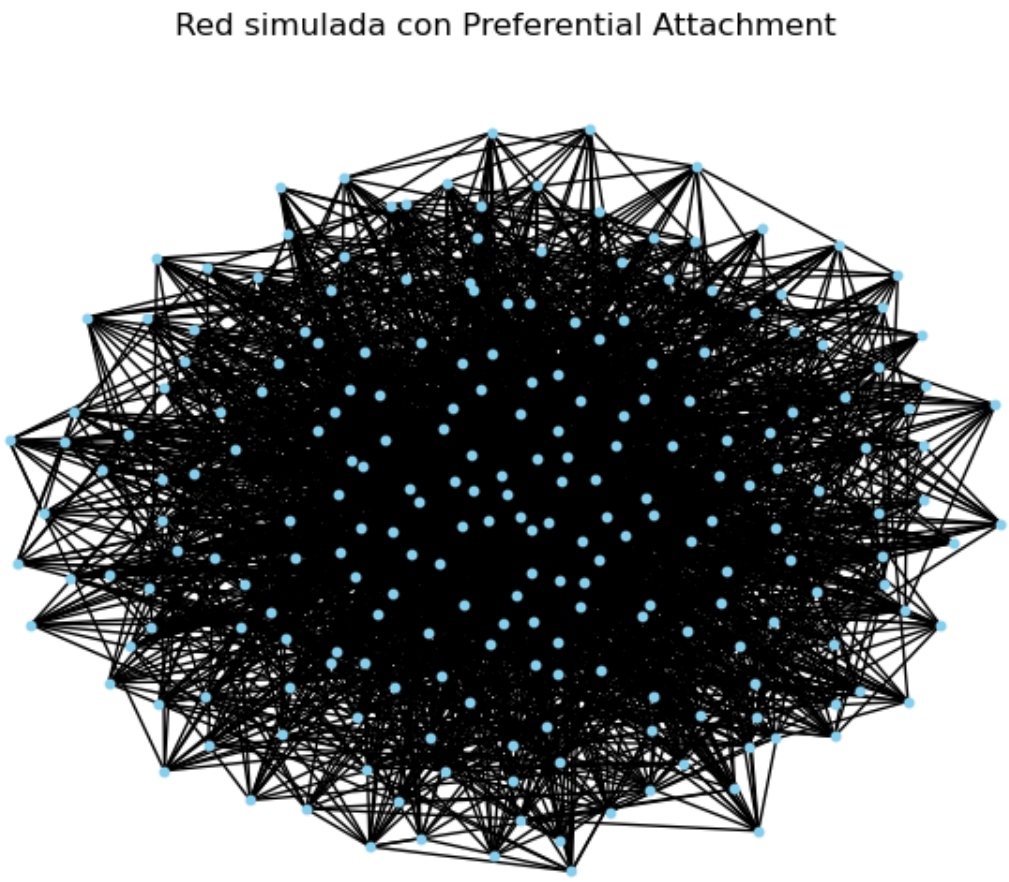


b. Obtener una simulación de un modelado de Preferential Attachment (ley de potencias) que corresponda a los parámetros de esta red.

In [31]:
```
n = G.number_of_nodes()
m = round(mean_degree / 2)

# Documentación del método:
# https://networkx.org/documentation/stable/reference/generated/networkx.genera
# Genera un grafo usando preferential attachment del tipo Barabasi Albert.
G_preferential_att = nx.barabasi_albert_graph(n, m)
```

In [32]:
```
pos = nx.spring_layout(G_preferential_att)
nx.draw(G_preferential_att, pos, node_size=10, node_color='skyblue')
plt.title("Red simulada con Preferential Attachment")
plt.show()

mean_degree_preferential_att = sum(dict(G_preferential_att.degree()).values())
clust_preferential_att = nx.average_clustering(G_preferential_att)
diameter_preferential_att = nx.diameter(G_preferential_att)
shortest_path_pref = nx.average_shortest_path_length(G_preferential_att)
connected_components_preferential_att = len(list(nx.connected_components(G_pref

print("Resultados".center(70, "-"))
print("Calculo".center(20) + "Red Real".center(20) + "Preferential Attachment"
print("Coef. Clustering".ljust(20) + "{:.2f}".format(clustering_coef).rjust(20)
print("Diametros".ljust(20) + f"{diameter}".rjust(20) + f"{diameter_preferentia
print("Distancia Promedio".ljust(20) + "{:.2f}".format(shortest_path_dist).rjus
```

```
print("Grado Promedio".ljust(20) + "{:.2f}".format(mean_degree).rjust(20) + "{
print("Componentes Conexas".ljust(20) + f"{connected_components}".rjust(20) + 
```

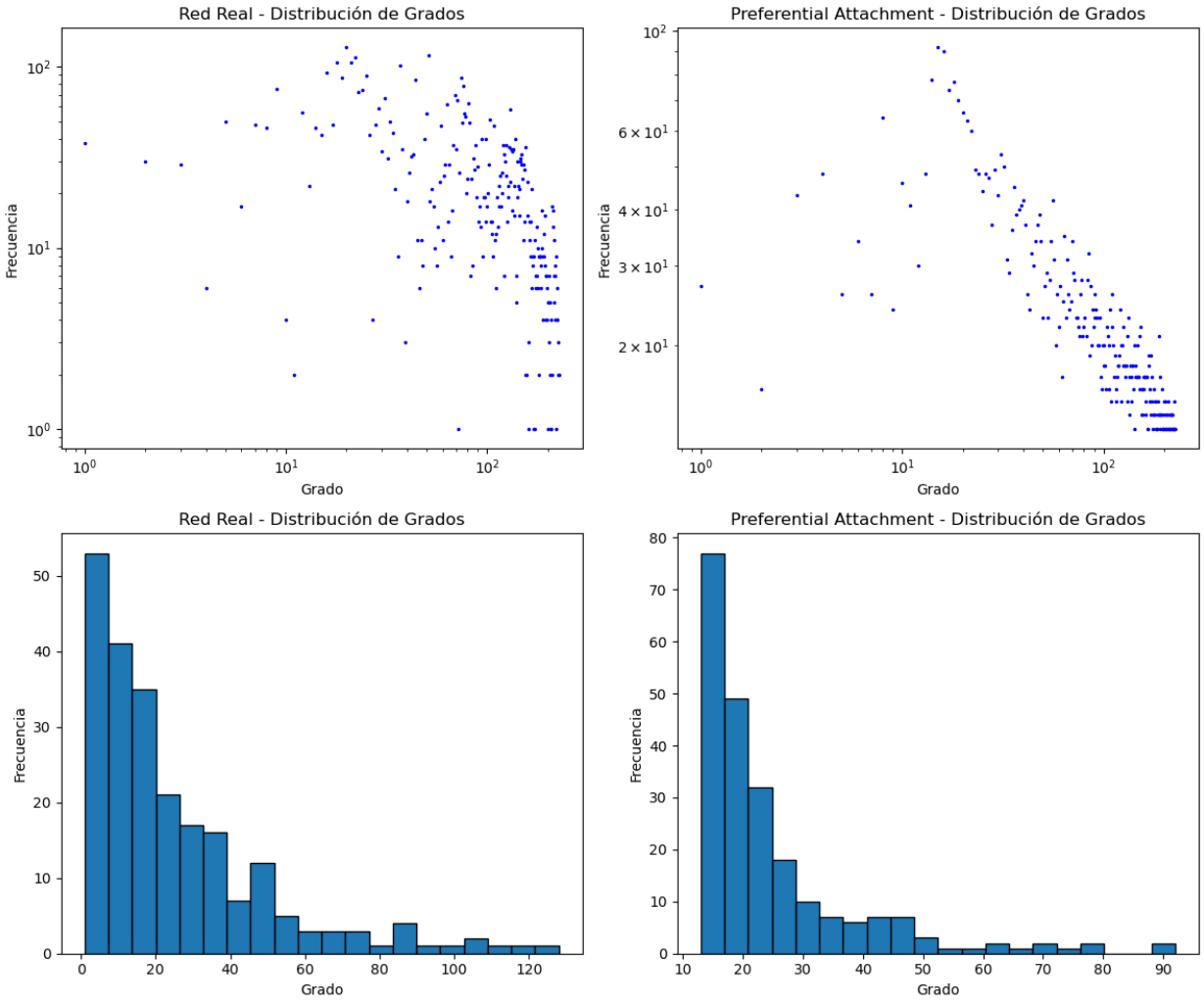



```
------------------------------Resultados------------------------------
      Calculo             Red Real      Preferential Attachment
Coef. Clustering                0.66                 0.19
Diametros                          4                    3
Distancia Promedio              2.24                 1.98
Grado Promedio                 25.02                24.52
Componentes Conexas                1                    1
```

Utilizando Preferential Attachment, el Coeficiente de Clustering no mejoró significativamente. Sin embargo, tal como se ve en las siguientes figuras, la distribución de los grados mejoró, aunque con un corrimiento horizontal sobre el grado mínimo.

In [33]:
```
degrees = [degree for node, degree in G.degree()]
degrees_preferential_att = [degree for node, degree in G_preferential_att.degre

fig, axes = plt.subplots(2, 2, figsize=(12, 10))

axes[0, 0].loglog(degrees, 'b.', markersize=3)
axes[0, 0].set_title("Red Real - Distribución de Grados")
axes[0, 0].set_xlabel("Grado")
axes[0, 0].set_ylabel("Frecuencia")

axes[0, 1].loglog(degrees_preferential_att, 'b.', markersize=3)
axes[0, 1].set_title("Preferential Attachment - Distribución de Grados")
axes[0, 1].set_xlabel("Grado")
axes[0, 1].set_ylabel("Frecuencia")
```

```
axes[1, 0].hist(degrees, bins=20, edgecolor='k')
axes[1, 0].set_title("Red Real - Distribución de Grados")
axes[1, 0].set_xlabel("Grado")
axes[1, 0].set_ylabel("Frecuencia")

axes[1, 1].hist(degrees_preferential_att, bins=20, edgecolor='k')
axes[1, 1].set_title("Preferential Attachment - Distribución de Grados")
axes[1, 1].set_xlabel("Grado")
axes[1, 1].set_ylabel("Frecuencia")

plt.tight_layout()
plt.show()
```



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.

In [34]:
```
import random
import math

def anon_walk(G, length):
    v = random.choice(list(G.nodes))
    anon_dict = {}
    anon_dict[v] = "0"
    anon = "0"
```

```
    while len(anon) < length:
        v = random.choice(list(G.neighbors(v)))
        if v in anon_dict:
            anon += anon_dict[v]
        else:
            anon += str(len(anon_dict))
            anon_dict[v] = str(len(anon_dict))
    return anon
```

In [35]:
```
length = 5
nu = 52
epsilon = 0.01
delta = 0.01
m = math.ceil((2 / (epsilon**2) * (math.log(2**nu - 2) - math.log(delta))))
```

In [36]:
```
def get_anon_vector(G, m, length):
    anons = defaultdict(int)
    for i in range(m):
        walk = anon_walk(G, length)
        anons[walk] += 1
    return dict(anons)
```

In [37]:
```
G_vect = get_anon_vector(G, m, length)
G_vect_pref = get_anon_vector(G_preferential_att, m, length)
G_vect_erdos = get_anon_vector(G_erdos, m, length)
```

In [38]:
```
def norm(anon_vect):
    return math.sqrt(sum([item ** 2 for item in anon_vect.values()]))

def cosine_dist(anon_vect_1, anon_vect_2):
    cumm = 0
    for k, v in anon_vect_1.items():
        if k in anon_vect_2:
            cumm += anon_vect_2[k] * v
    return cumm / (norm(anon_vect_1) * norm(anon_vect_2))
```

In [39]:
```
print(f"Distancia entre G real y Erdos: {cosine_dist(G_vect, G_vect_pref)}")
print(f"Distancia entre G real y Preferential Attachment: {cosine_dist(G_vect,
```

```
Distancia entre G real y Erdos: 0.9998371295738322
Distancia entre G real y Preferential Attachment: 0.9998089810451462
```

Luego de haber corrido el algoritmo múltiples veces con distintos valores de length, no pude sacar resultados concluyentes entre las simulaciones. Por lo general suele tener más cercanía Preferential Attachment, pero el resultado no es tan significativo.

Ambas tienen una distancia coseno muy similar a la de la red real, principalmente porque los anonymous walks suelen ser de secuencias con todos nodos diferentes (secuencia 1-2-3-4-5). Esto genera que el ángulo entre los vectores sea muy pequeño (la magnitud de su componente 1-2-3-4-5 es muy alta).

Ejercicio 5

a. Calcular los motifs de hasta 5 nodos de una subred definida en el punto 2.b.

In [21]:
```
def subnet_community(G, community):
    G_subnet = G.copy()
    nodes_to_remove = [node for node in G_subnet.nodes() if node not in communi
    G_subnet.remove_nodes_from(nodes_to_remove)
    return G_subnet
```

In [22]:
```
one_community = communities[0]
G_subnet = subnet_community(G, one_community)
```

In [23]:
```
G_subnet = G.copy()
one_community = communities[0]
nodes_to_remove = [node for node in G_subnet.nodes() if node not in one_communi
G_subnet.remove_nodes_from(nodes_to_remove)
```

In [26]:
```
G_subnet.number_of_edges()
```

Out[26]:
```
639
```

In [27]:
```
from motifs.calculos import calcular_motifs

MAX_MOTIFS = 5

motifs = calcular_motifs(G_subnet, MAX_MOTIFS)
```

In [28]:
```
motifs
```

Out[28]:
```
array([  7030,   4835,  14087,  27818,   2444,  62375,  48392,  24720,
        16550,  65322,  75425, 100362,  36726, 231182,   1247,  25224,
       488473,  30303, 115053,   2310,  28470, 161381, 374557, 227832,
        20681, 513184,  33980, 283192,  92115])
```

Este último cálculo con 639 aristas y 50 nodos tardó más de 2 horas en ejecutarse en mi computadora (no está en muy buen estado). Para el cálculo del promedio y desvío estándar tuve a mi computadora una noche trabajando para 15 iteraciones, y se me interrumpió el kernel de jupyter. Lo mismo me pasó al intentar correrlo durante el día.

Es una lástima porque el análisis hubiera sido más rico con esa información, pero no pude hacerlo. Como solución para poder entregar algo, uso una sub comunidad de una comunidad. Con este, los algoritmos corren sumamente rápido porque son de muchos menos nodos, pero al mismo tiempo, los resultados son mucho menos representativos.

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.

In [61]:
```
# Calculo los motifs de la sub sub comunidad.

sub_communities = nx.community.louvain_communities(G_subnet, seed=42)
one_community = list(sub_communities)[0]
```

```
G_subsubnet = G_subnet.copy()
nodes_to_remove = [node for node in G_subnet.nodes() if node not in one_communi
G_subsubnet.remove_nodes_from(nodes_to_remove)
```

In [63]:
```
motifs_subsubnet = calcular_motifs(G_subsubnet, MAX_MOTIFS)
motifs_subsubnet
```

Out[63]:
```
array([ 44,  54,  18,  14,   0,  88,  58,  56,   0,  12,   0,   3,   0,
        35,   0,   0,  30,   0,  55,   0,   0,  10, 165,  44,   0,  83,
         0,  68,  36])
```

In [64]:
```
from motifs.calculos import motif_grafo_eleatorios, significance_profile

n_subsubnet = G_subsubnet.number_of_nodes()
mean_degree_subnet = sum(dict(G_subsubnet.degree()).values()) / len(G_subsubnet
p_subsubnet = mean_degree_subnet / (n_subnet - 1)

# Uso como red Baseline Erdos Renyi.

N_rand_prom, N_rand_stds = motif_grafo_eleatorios(lambda: nx.erdos_renyi_graph(
```

```
Iteracion 1
Iteracion 2; anterior: 0.48 segs
Iteracion 3; anterior: 0.46 segs
Iteracion 4; anterior: 0.49 segs
Iteracion 5; anterior: 0.52 segs
Iteracion 6; anterior: 0.52 segs
Iteracion 7; anterior: 0.46 segs
Iteracion 8; anterior: 0.46 segs
Iteracion 9; anterior: 0.45 segs
Iteracion 10; anterior: 0.48 segs
Iteracion 11; anterior: 0.46 segs
Iteracion 12; anterior: 0.47 segs
Iteracion 13; anterior: 0.48 segs
Iteracion 14; anterior: 0.46 segs
Iteracion 15; anterior: 0.46 segs
```

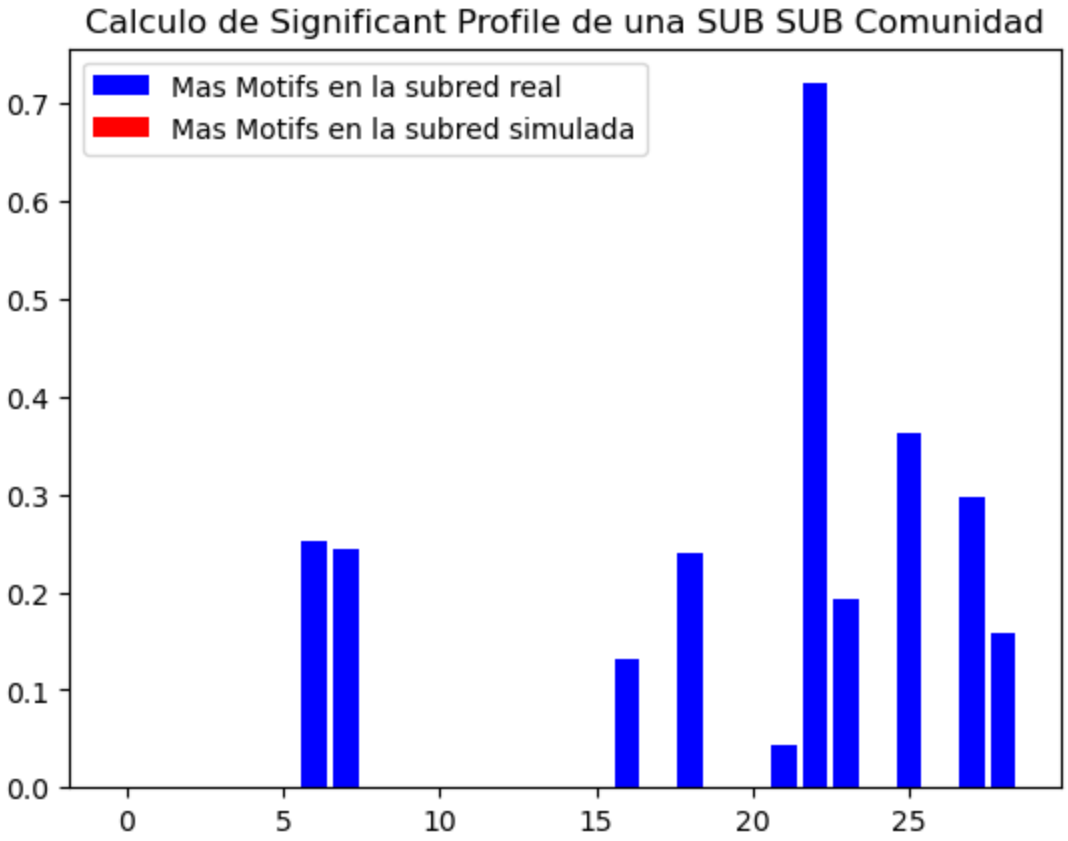
In [65]:
```
N_real = motifs_subsubnet

SP = significance_profile(N_real, N_rand_prom, N_rand_stds)
```

In [69]:
```
SP
```

Out[69]:
```
array([ 2.52170221e-05,  5.85570003e-04,  8.82526964e-06,  6.34366567e-06,
       -1.71066497e-06,  2.82162101e-04,  2.53144742e-01,  2.44415613e-01,
       -2.12261361e-06,  3.45359273e-06, -1.28909879e-06,  1.63089114e-05,
       -1.67277187e-06,  3.04424036e-04,  0.00000000e+00, -1.61139168e-06,
        1.30936936e-01,  0.00000000e+00,  2.40051049e-01,  0.00000000e+00,
       -1.16554442e-06,  4.36456452e-02,  7.20153146e-01,  1.92040839e-01,
        0.00000000e+00,  3.62258855e-01,  0.00000000e+00,  2.96790387e-01,
        1.57124323e-01])
```

In [75]:
```
positive_values = [val if val >= 0 else 0 for val in SP]
negative_values = [val if val < 0 else 0 for val in SP]

x = range(len(SP))
plt.bar(x, positive_values, color='b', label='Mas Motifs en la subred real')
plt.bar(x, negative_values, color='r', label='Mas Motifs en la subred simulada

plt.legend()
```

```
plt.title("Calculo de Significant Profile de una SUB SUB Comunidad")
plt.show()
```

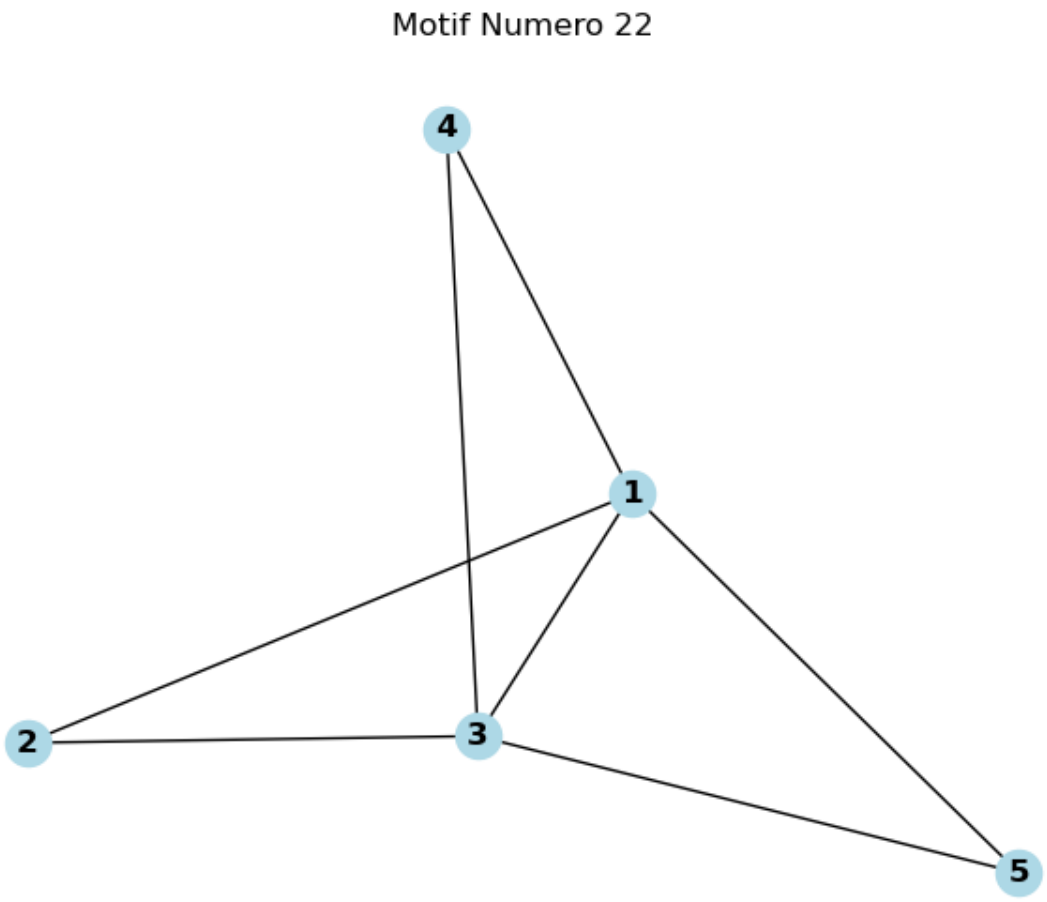


c. Intentar dar con una explicación del resultado obtenido en el punto anterior.

Las interpretaciones que puedo tomar aquí pueden no ser tan correctas, al haber usado una sub sub comunidad para los cálculos. Sin embargo, hay roles o comportamientos que se mantienen en las subredes existentes. Esto se muestra muy claramente en el ejercicio de roles.

En los últimos motifs (desde el motif 20 en adelante principalmente), el coeficiente de clustering promedio de estos motifs es alto, lo cual puede explicar el resultado obtenido. Tomo como ejemplo el motif 22 y calculo su coeficiente de clustering a continuación.

In [90]:
```
exampleG = nx.Graph()
exampleG_nodes = [1, 2, 3, 4, 5]
exampleG.add_nodes_from(exampleG_nodes)
exampleG.add_edge(1, 2)
exampleG.add_edge(2, 3)
exampleG.add_edge(3, 4)
exampleG.add_edge(4, 1)
exampleG.add_edge(5, 1)
exampleG.add_edge(5, 3)
exampleG.add_edge(1, 3)

nx.draw(exampleG, with_labels=True, node_color='lightblue', font_size=12, font_
plt.title("Motif Numero 22")
plt.show()
```



In [92]:
```
print(f"El coeficiente de clustering del motif 22 es {nx.average_clustering(exa
```

```
El coeficiente de clustering del motif 22 es 0.8
```

Como se puede ver, el coeficiente de clustering es sumamente alto. Para los últimos motifs, se encontrarían coeficientes de clustering similares. En el ejercicio de análisis de modelos, se observó que el coeficiente de clustering es mucho mayor en la red con respecto al de los modelos simulados (en este caso Erdös Renyi). En la red, y en general en las redes sociales, el coeficiente de clustering suele ser alto, debido a la gran probabilidad de generar triángulos. Este punto no es tan fuerte en el modelo utilizado para comparar.

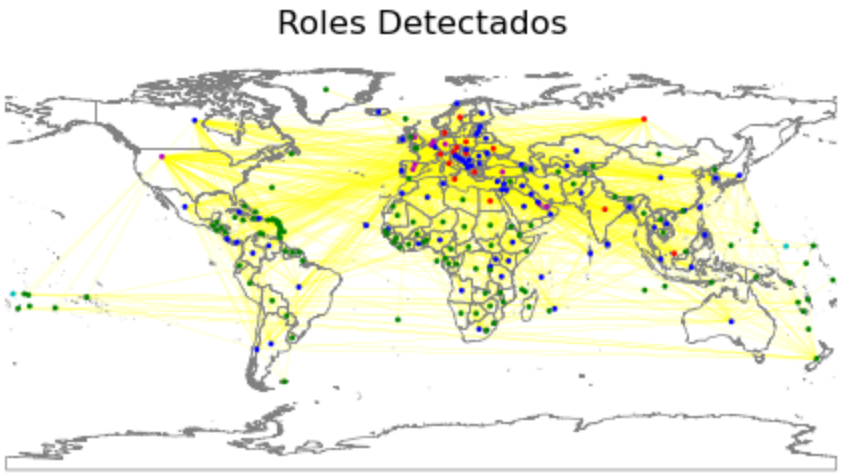
Ejercicio 6

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

In [111…
```
from graphrole import RoleExtractor, RecursiveFeatureExtractor

feature_extractor = RecursiveFeatureExtractor(G)
features = feature_extractor.extract_features()

role_extractor = RoleExtractor(n_roles=None)
role_extractor.extract_role_factors(features)
```

In [112…
```
roles = role_extractor.roles

role_colors = {
    'role_0': 'r',
    'role_1': 'g',
    'role_2': 'b',
    'role_3': 'c',
    'role_4': 'm',
    'role_5': 'y',
    'role_6': 'k'
}
node_colors = [role_colors[roles[country]] for country in G.nodes()]
```

In [113…
```
world = gdf.boundary.plot(color='gray', linewidth=0.5)
pos = {row["NAME_EN"]: (row["x"], row["y"]) for _,row in df_more_data.iterrows(
nx.draw(G, pos, with_labels=False,node_color=node_colors, width= 0.1, node_size

plt.title('Roles Detectados')
plt.show()
```

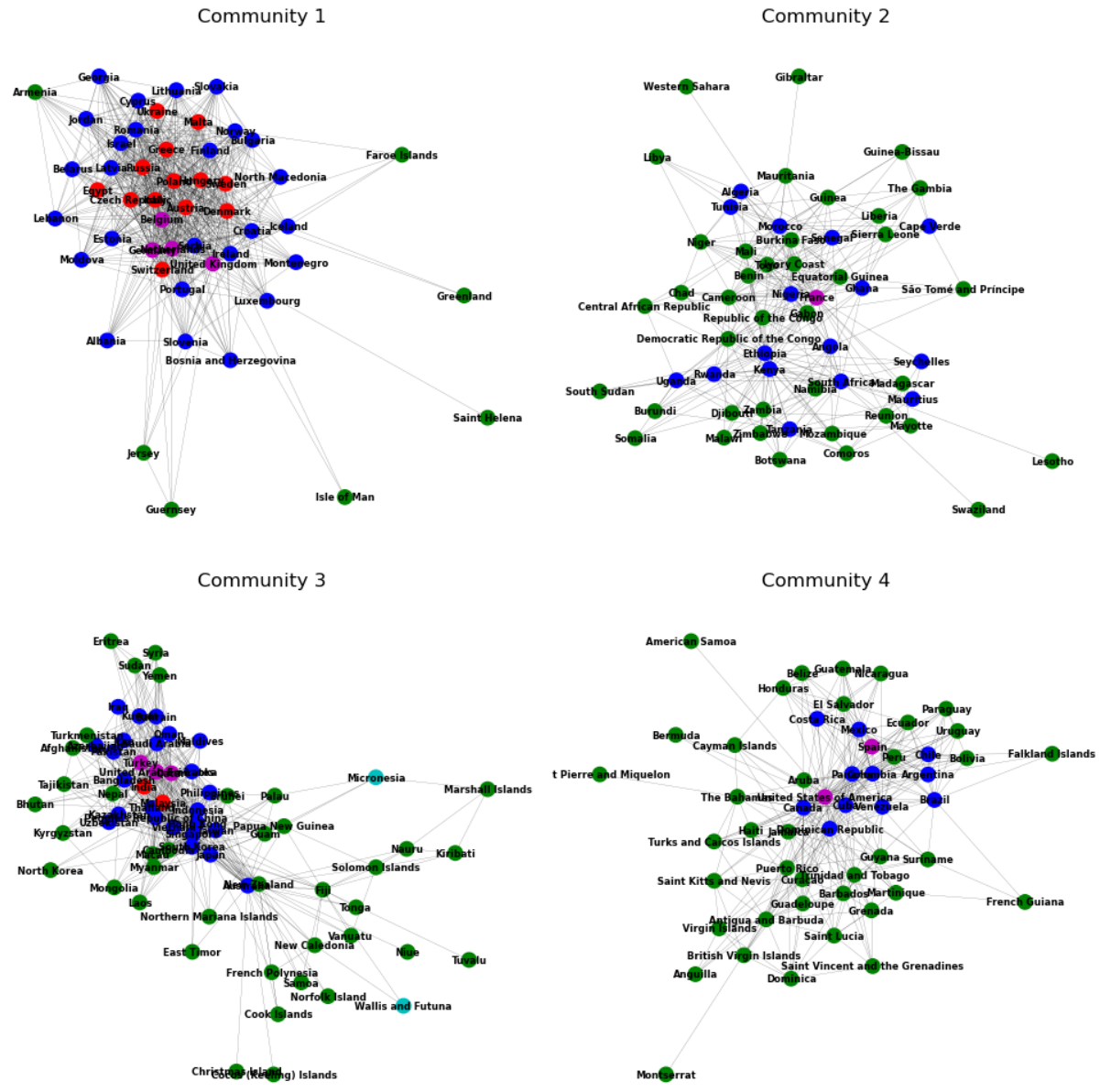


De este gráfico se puede observar un poco de información, pero es muy dificil interpetarlo. A simple vista, los nodos más centrales de la red están en europa / Estados Unidos con los colores rojo y violeta. Los más periféricos, en verde. Y en un punto medio, los nodos azules.

Dado que esto no se puede ver tan claramente en este mapa, divido el gráfico por comunidades, para poder hacer zoom en el comportamiento de cada comunidad.

In [122…
```
import matplotlib.pyplot as plt

def draw_community_roles(G_subnet, roles, ax):
    c = [role_colors[roles[country]] for country in G_subnet.nodes()]
    nx.draw(G_subnet, with_labels=True, node_color=c, width=0.1, node_size=80,
            edge_color='k', font_size=6, font_color='black', font_weight='bold

num_communities = len(communities)
num_cols = 2
num_rows = (num_communities + num_cols - 1) // num_cols

fig, axes = plt.subplots(num_rows, num_cols, figsize=(10, num_rows * 5))
for i, community in enumerate(communities):
```

```
    row = i // num_cols
    col = i % num_cols
    ax = axes[row, col]
    G_community = subnet_community(G, community)
    draw_community_roles(G_community, roles, ax)
    ax.set_title(f"Community {i + 1}")

for i in range(num_communities, num_rows * num_cols):
    row = i // num_cols
    col = i % num_cols
    fig.delaxes(axes[row, col])

plt.tight_layout()
plt.show()
```



En estos gráficos se pueden ver los roles por color, para cada país, separados por comunidad. Vease cómo los nodos rojos son los más centrales, y aparecen en su mayoría en la comunidad que se corresponde con Europa (comunidad 1).

Para la comunidad de América, que quizas es la más conocida por mi familiaridad con la gran mayoría de los paises, queda muy claro el rol de los nodos según el grado de centralidad en la comunidad, y en la red.

Los países de América Central, e islas como las Malvinas, son más periféricas. Además, conectarlas con otras comunidades es dificil sin hacer una escala por nodos más centrales.

En segunda escala se encuentras países más grandes, como Argentina, Brasil y México, que tienen más vuelos, y más llegada al mundo.

En violeta, y con mayor centralidad, se observa a Estados Unidos. También figura España en la comunidad de América, aunque no sea lo correcto.

Ejercicio 7

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

In [58]:
```
bridges = list(nx.bridges(G))
print("Los puentes globales son los siguientes:")
bridges
```

Out[58]:
```
Los puentes globales son los siguientes:
[('Fiji', 'Tuvalu'),
 ('United States of America', 'American Samoa'),
 ('United Kingdom', 'Saint Helena'),
 ('Canada', 'Saint Pierre and Miquelon'),
 ('Antigua and Barbuda', 'Montserrat'),
 ('New Zealand', 'Niue'),
 ('South Africa', 'Lesotho'),
 ('South Africa', 'Swaziland')]
```

Algo esperado de todos estos puentes, es que todos contienen a un nodo cuyo grado es 1. Por lo tanto, el puente es la única conexión de un país / isla, con el mundo.

In [60]:
```
grade_one = True

for a, b in bridges:
    if G.degree[a] != 1 and G.degree[b] != 1:
        grade_one = False
        print(f"El siguiente puente desconecta a la red en dos componentes de r

if grade_one:
    print("Todos los puentes globales desconectan a un solo pais.")
```

```
Todos los puentes desconectan a un solo pais.
```

In [71]:
```
local_bridges = list(nx.local_bridges(G))
local_bridges = list(filter(lambda x: x[2] != float('inf'), local_bridges))
print("Los puentes locales (no globales) son:")
local_bridges
```

Out[71]:
```
Los puentes locales (no globales) son:
[('Papua New Guinea', 'Micronesia', 3), ('Micronesia', 'Marshall Islands',
3)]
```

Solamente hay dos puentes locales (que no son globales). Esto también da una idea de la gran cantidad de triángulos que hay presentes en la red.