azqyetaat

November 30, 2024

[20]:
```
import scipy.io
math_path = r"C:\Users\gia19\OneDrive\Documentos\RepositorioCacahuate\Parcial␣
 ↪2\Coactivation_matrix.mat"
contents = scipy.io.loadmat(math_path)
coactivation_matrix = contents["Coactivation_matrix"]
coord = contents["Coord"]
coactivation_matrix.shape, coord.shape
```

[20]:
```
((638, 638), (638, 3))
```

0.0.1 Ejercicio 1

Definir grafos con la matriz estableciendo umbrales de coactivación de 0.8, 0.9 y 1 y graficar cada grafo. Añadir las coordenadas tridimensionales

[24]:
```
import networkx as nx
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np


def haciendo_grafo(matrix, coords, threshold, ax):
    adjacency_matrix = (matrix>=threshold).astype(int)

    G = nx.from_numpy_array(adjacency_matrix)
    # Graficar en 3D
    ax.set_title(f"Threshold: {threshold}")
    ax.axis("off")
    for edge in G.edges():
        x_vals = [coords[edge[0], 0], coords[edge[1], 0]]
        y_vals = [coords[edge[0], 1], coords[edge[1], 1]]
        z_vals = [coords[edge[0], 2], coords[edge[1], 2]]
        ax.plot(x_vals, y_vals, z_vals, c='g', alpha=0.5, linewidth=0.7)

    ax.scatter(coords[:, 0], coords[:, 1], coords[:, 2], c='r', s=10)

fig = plt.figure(figsize=(18, 6))
```

```
thresholds = [0.08, 0.09, 0.1]

for i, threshold in enumerate(thresholds, 1):
    ax = fig.add_subplot(1, 3, i, projection='3d')
    haciendo_grafo(coactivation_matrix, coord, threshold, ax)

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

0.0.2 Ejercicio 2

Con uno de los grafos en el punto uno con umbral 0.9, generar una animación donde se haga girar 360° el grafo del cerebro para visualizar las conexiones establecidas.

[29]:
```
from matplotlib.animation import FuncAnimation

def update_rotation(frame, graph, coords, ax):
    ax.cla()  # Limpiar el eje
    ax.set_title("Grafo 360°")
    ax.axis("off")
    for edge in graph.edges():
        x_vals = [coords[edge[0], 0], coords[edge[1], 0]]
        y_vals = [coords[edge[0], 1], coords[edge[1], 1]]
        z_vals = [coords[edge[0], 2], coords[edge[1], 2]]
        ax.plot(x_vals, y_vals, z_vals, c='g', alpha=0.5, linewidth=0.7)
    ax.scatter(coords[:, 0], coords[:, 1], coords[:, 2], c='r', s=10)
    ax.view_init(30, frame)  # Cambiar el ángulo de vista

# Grafo para el umbral de 0.09
adjacency_matrix_09 = (coactivation_matrix >= 0.09).astype(int)
graph_09 = nx.from_numpy_array(adjacency_matrix_09)

fig = plt.figure(figsize=(6,6))
```

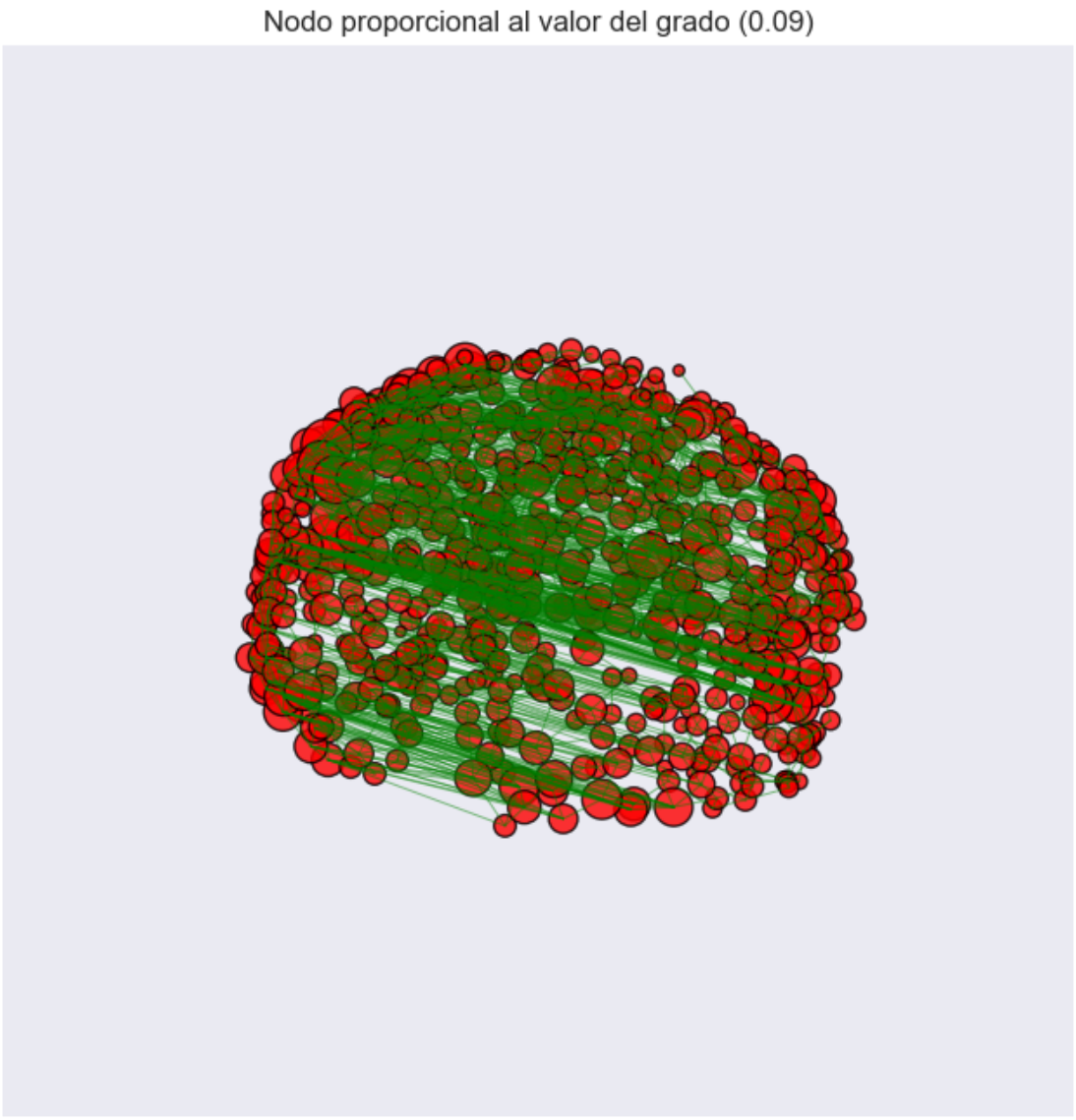
```
ax = fig.add_subplot(111, projection='3d')

anim = FuncAnimation(fig, update_rotation, frames=360, fargs=(graph_09, coord,␣
↪ax), interval=50)
```

0.0.3 Ejercicio 3

Encontrar los hubs del grafo, y establecer el tamaño del nodo proporcional al valor del grado.

[32]:
```
degrees = dict(graph_09.degree())

max_degree = max(degrees.values())
node_sizes = [500 * (deg / max_degree) for deg in degrees.values()]

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
```

```python
# Para las aristas:
for edge in graph_09.edges():
    x_vals = [coord[edge[0], 0], coord[edge[1], 0]]
    y_vals = [coord[edge[0], 1], coord[edge[1], 1]]
    z_vals = [coord[edge[0], 2], coord[edge[1], 2]]
    ax.plot(x_vals, y_vals, z_vals, c='g', alpha=0.5, linewidth=0.7)

ax.scatter(
    coord[:, 0], coord[:, 1], coord[:, 2],
    c='r', s=node_sizes, alpha=0.8, edgecolors='k'
)

ax.set_title("Nodo proporcional al valor del grado (0.09)")
ax.axis("off")

plt.show()
```



0.0.4 Ejercicio 4

En función de la matriz de emparejamiento (correlación de la matriz de adyacencia), establecer una partición de los nodos en módulos. Escoger el número de módulos que creas conveniente y justificar por qué escogiste ese número.

[33]:
```
from sklearn.cluster import SpectralClustering

graph = nx.from_numpy_array(coactivation_matrix)
# Rango de 2 a 10 módulos, esto evaluando la modularidad, para saber que tan␣
 ↪bien separados están los módulos dentro de una red, estos números aseguran␣
 ↪que los módulos capturan una estructura coherente en la red sin fragmentarla
```

```
range_clusters = range(2, 11)
modularity_scores = []
best_partition = None

# Calcular con diferentes números de módulos
for n_clusters in range_clusters:
    clustering = SpectralClustering(
        n_clusters=n_clusters,
        affinity='precomputed',
        random_state=42
    ).fit(coactivation_matrix)

    labels = clustering.labels_

    partition_spectral = {i: label for i, label in enumerate(labels)}
    modularity = nx.algorithms.community.quality.modularity(graph, [
        [node for node, cluster in partition_spectral.items() if cluster == c]
        for c in np.unique(labels)
    ])

    modularity_scores.append(modularity)

    if not best_partition or modularity > max(modularity_scores[:-1]):
        best_partition = partition_spectral

optimal_clusters = range_clusters[np.argmax(modularity_scores)]

optimal_clusters, max(modularity_scores)
```

[33]: (8, 0.4448068669431869)

0.0.5 Ejercicio 5

Determinar el conjunto del RichClub y discutir las implicaciones anatomica y funcionales de este grupo de nodos

[2]:
```
import scipy.io as sio
import networkx as nx
import numpy as np

# Cargar los datos
mat_data = sio.loadmat(r'C:
 ↪\Users\gia19\OneDrive\Documentos\RepositorioCacahuate\Parcial␣
 ↪2\Coactivation_matrix.mat')
coactivation_matrix = mat_data['Coactivation_matrix']

# Crear el grafo
```

```
G = nx.from_numpy_array(coactivation_matrix)

# Identificar el conjunto de Rich Club
rich_club_nodes = [node for node, degree in G.degree() if degree > np.mean([deg␣
 ↪for _, deg in G.degree()])]
print(f"Nodos en el Rich Club: {len(rich_club_nodes)}")
print(f"Algunos nodos del Rich Club: {rich_club_nodes[:10]}")
```

```
Nodos en el Rich Club: 255
Algunos nodos del Rich Club: [6, 7, 11, 16, 18, 19, 20, 22, 37, 38]
```

0.0.6 Discusión

Las áreas del cerebro con mayor numero de conexiones, conocidas como hubs, suelen tener un apepl central en la integración y transmisión de información entre distintas regiones. Entre algunas de estas estructuras relevantes se enceuntran: - la corteza prefrontal medial (mPFC): que cumple su función en la toma de desiciones, planificación y regulación emocional - Precúneo: procesa imágenes mentales y se encarga de la autoconsciencia - Corteza cingulada posterior (PCC): participa en la memoria y en la regulación de las redes por defecto. - Corteza parietal superior (SPL): permite la atenci”n y la percepción espacial - Tálamo: integración sensorial y coordinación motora - Cortezas visual primaria y secundaria (V1, V2): procesamiento de estimulos visuales

0.0.7 Ejercicio 6

Supongamos que eliminamos los nodos del RichClub, describir cómo cambian las propiedades topológicas del grafo, hacer comparativas del grado, coeficiente de cluster, coeficiente de mundo y pequeño y las medidas de centralaidad (cercanía e intermediación)

[52]:
```
import networkx as nx
import scipy.io as sio
import numpy as np
mat_data = sio.loadmat(r'C:
 ↪\Users\gia19\OneDrive\Documentos\RepositorioCacahuate\Parcial␣
 ↪2\Coactivation_matrix.mat')
coactivation_matrix = mat_data['Coactivation_matrix']
G = nx.from_numpy_array(coactivation_matrix)
rich_club_nodes = [node for node, degree in G.degree() if degree > np.mean([deg␣
 ↪for _, deg in G.degree()])]

# Crear un grafo sin los nodos del Rich Club

G_without_rich_club = G.copy()
G_without_rich_club.remove_nodes_from(rich_club_nodes)

# Función para calcular propiedades topológicas
def calculate_graph_properties(G):
    degree = np.mean([deg for _, deg in G.degree()])
    clustering_coeff = nx.average_clustering(G)
```

```
    small_world_coeff = nx.sigma(G) if nx.is_connected(G) else None
    closeness_centrality = np.mean(list(nx.closeness_centrality(G).values()))
    betweenness_centrality = np.mean(list(nx.betweenness_centrality(G).
 ↪values()))

    return {
        "degree": degree,
        "clustering_coefficient": clustering_coeff,
        "small_world_coefficient": small_world_coeff,
        "closeness_centrality": closeness_centrality,
        "betweenness_centrality": betweenness_centrality,
    }
# Comparar propiedades
original_properties = calculate_graph_properties(G)
rich_club_removed_properties = calculate_graph_properties(G_without_rich_club)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[52], line 30
     22     return {
     23         "degree": degree,
     24         "clustering_coefficient": clustering_coeff,
   (…)
     27         "betweenness_centrality": betweenness_centrality,
     28     }
     29 # Comparar propiedades
---> 30 original_properties = calculate_graph_properties(G)
     31 rich_club_removed_properties =␣
 ↪calculate_graph_properties(G_without_rich_club)

Cell In[52], line 18, in calculate_graph_properties(G)
     16 degree = np.mean([deg for _, deg in G.degree()])
     17 clustering_coeff = nx.average_clustering(G)
---> 18 small_world_coeff = nx.sigma(G) if nx.is_connected(G) else None
     19 closeness_centrality = np.mean(list(nx.closeness_centrality(G).values() )
     20 betweenness_centrality = np.mean(list(nx.betweenness_centrality(G).
 ↪values()))

File <class 'networkx.utils.decorators.argmap'> compilation 36:6, in␣
 ↪argmap_sigma_30(G, niter, nrand, seed, backend, **backend_kwargs)
      4 import inspect
      5 import itertools
----> 6 import re
      7 import warnings
      8 from collections import defaultdict
```

```
File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi:
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith:s\smallworld
 ↪py:301, in sigma(G, niter, nrand, seed)
    299 randMetrics = {"C": [], "L": []}
    300 for i in range(nrand):
--> 301     Gr = random_reference(G, niter=niter, seed=seed)
    302     randMetrics["C"].append(nx.transitivity(Gr))
    303     randMetrics["L"].append(nx.average_shortest_path_length(Gr))

File <class 'networkx.utils.decorators.argmap'> compilation 43:6, in␣
 ↪argmap_random_reference_37(G, niter, connectivity, seed, backend,␣
 ↪**backend_kwargs)
      4 import inspect
      5 import itertools
----> 6 import re
      7 import warnings
      8 from collections import defaultdict

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi:
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith:s\smallworld
 ↪py:109, in random_reference(G, niter, connectivity, seed)
    106 G.remove_edge(c, d)
    108 # Check if the graph is still connected
--> 109 if connectivity and local_conn(G, a, b) == 0:
    110     # Not connected, revert the swap
```

```
    111     G.remove_edge(a, d)
    112     G.remove_edge(c, b)

File <class 'networkx.utils.decorators.argmap'> compilation 51:3, in␣
↪argmap_local_edge_connectivity_48(G, s, t, flow_func, auxiliary, residual,␣
↪cutoff, backend, **backend_kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
↪messages.
    972 backend_name = backend

File␣
↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\connectivit
↪py:643, in local_edge_connectivity(G, s, t, flow_func, auxiliary, residual,␣
↪cutoff)
    640 if flow_func is shortest_augmenting_path:
    641     kwargs["two_phase"] = True
--> 643 return nx.maximum_flow_value(H, s, t, **kwargs)

File <class 'networkx.utils.decorators.argmap'> compilation 59:3, in␣
↪argmap_maximum_flow_value_56(flowG, _s, _t, capacity, flow_func, backend,␣
↪**kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
```

```
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith␣s\flow\maxflo
 ↪py:302, in maximum_flow_value(flowG, _s, _t, capacity, flow_func, **kwargs)
    299 if not callable(flow_func):
    300     raise nx.NetworkXError("flow_func has to be callable.")
--> 302 R =␣
 ↪flow_func(flowG, _s, _t, capacity=capacity, value_only=True, **kwargs)
    304 return R.graph["flow_value"]

File <class 'networkx.utils.decorators.argmap'> compilation 63:3, in␣
 ↪argmap_edmonds_karp_60(G, s, t, capacity, residual, value_only, cutoff,␣
 ↪backend, **backend_kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith␣s\flow\edmond
 ↪py:238, in edmonds_karp(G, s, t, capacity, residual, value_only, cutoff)
    120 @nx._dispatchable(edge_attrs={"capacity": float("inf")},␣
 ↪returns_graph=True)
    121 def edmonds_karp(
    122     G, s, t, capacity="capacity", residual=None, value_only=False,␣
 ↪cutoff=None
    123 ):
    124     """Find a maximum single-commodity flow using the Edmonds-Karp␣
 ↪algorithm.
    125
    126     This function returns the residual network resulting after computin
   (…)
    236
```

```
    237     """
--> 238     R = edmonds_karp_impl(G, s, t, capacity, residual, cutoff)
    239     R.graph["algorithm"] = "edmonds_karp"
    240     nx._clear_cache(R)

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorithms\flow\edmond
 ↪py:104, in edmonds_karp_impl(G, s, t, capacity, residual, cutoff)
    101     raise nx.NetworkXError("source and sink are the same node")
    103 if residual is None:
--> 104     R = build_residual_network(G, capacity)
    105 else:
    106     R = residual

File <class 'networkx.utils.decorators.argmap'> compilation 67:3, in␣
 ↪argmap_build_residual_network_64(G, capacity, backend, **backend_kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\backends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for this
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorithms\flow\utils
 ↪py:137, in build_residual_network(G, capacity)
    135 for u, v, attr in edge_list:
    136     r = min(attr.get(capacity, inf), inf)
--> 137     if not R.has_edge(u, v):
    138         # Both (u, v) and (v, u) must be present in the residual
    139         # network.
    140         R.add_edge(u, v, capacity=r)
    141         R.add_edge(v, u, capacity=0)

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\classes\graph.
 ↪py:1292, in Graph.has_edge(self, u, v)
   1289     else:
```

```
   1290         raise NetworkXError("update needs nodes or edges input")
-> 1292 def has_edge(self, u, v):
   1293     """Returns True if the edge (u, v) is in the graph.
   1294
   1295     This is the same as `v in G[u]` without KeyError exceptions.
   (…)
   1326
   1327     """
   1328     try:

KeyboardInterrupt:
```

0.0.8 Comentario

Este codigo señala las diferencias topológicas entre el grafo original y al que le fue removido el RichClub, sin embargo parece que el procesamiento de iterar sobre los 638 nodos es pesado (mi computadora llevaba 4hrs cargando y aun asi no mostraba el resultado:( ) sin embargo considero que es correcto

0.0.9 Discusión

Cuando se elimina el conjunto de RichClub, las propiedades topologicas se ven significativamente afectadas de la siguiente manera: - El coeficiente de cluster disminuye, lo que significa que hay menos enlaces entre vecinos y como resultado la red pierde su "densidad" local, disminuyendo la tendencia a la formacion de clusteres - La red se vuelve menos eficiente en cuanto a la conectividad local, ya que aumenta la distancia promedio entre nodos, lo que disminuye el coeficiente de mundo pequeño (aumento del numero de pasos necesarios para conectar dos nodos no cercanos). - Los nodos tendran una mayor distancia, lo que disminuye el valor de centralidad de cercania, - Los nodos de RichClub suelen tener una alta centralidad de intermediacion, ya que tienen muchas conexiones y participan en una gran cantidad de caminos mas cortos entre nodos. Al eliminarlos, los nodos restantes deben depender de otros nodos para mantener la conectividad, y como resultado, la centralidad de intermediacion de los nodos puede redistribuirse y los nodos restantes que antes no eran centrales pueden aumentar su centrlidad de intermediacion.

0.0.10 Ejercicio 7

Quitar 10% 50% de los nodos con mayor medida de intermediación y describir cómo cambian las propiedades topológicas del grafo, hacer comparativas del grado, coeficiente de cluster, coeficiente de mundo pequeño y las medidas de centralidad

[56]:
```python
# Identificar los nodos con mayor intermediación
betweenness = nx.betweenness_centrality(G)
sorted_nodes = sorted(betweenness, key=betweenness.get, reverse=True)

# Eliminar el 10% y 50% de los nodos con mayor intermediación
porcentajes = [0.1, 0.5]
for p in porcentajes:
    num_remove = int(len(sorted_nodes) * p)
```

```
    nodes_to_remove = sorted_nodes[:num_remove]

    G_modified = G.copy()
    G_modified.remove_nodes_from(nodes_to_remove)

    modified_properties = calculate_graph_properties(G_modified)
    print(f"Propiedades al eliminar el {int(p*100)}% de nodos con mayor␣
↪intermediación:", modified_properties)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[56], line 14
     11 G_modified = G.copy()
     12 G_modified.remove_nodes_from(nodes_to_remove)
---> 14 modified_properties = calculate_graph_properties(G_modified)
     15 print(f"Propiedades al eliminar el {int(p*100)}% de nodos con mayor␣
 ↪intermediación:", modified_properties)

Cell In[55], line 14, in calculate_graph_properties(G_original)
     12 degree = np.mean([deg for _, deg in G_original.degree()])
     13 clustering_coeff = nx.average_clustering(G_original)
---> 14 small_world_coeff = nx.sigma(G_original) if nx.is_connected(G_original)
 ↪else None
     15 closeness_centrality = np.mean(list(nx.closeness_centrality(G_original)
 ↪values()))
     16 betweenness_centrality = np.mean(list(nx.
 ↪betweenness_centrality(G_original).values()))

File <class 'networkx.utils.decorators.argmap'> compilation 36:6, in␣
 ↪argmap_sigma_30(G, niter, nrand, seed, backend, **backend_kwargs)
      4 import inspect
      5 import itertools
----> 6 import re
      7 import warnings
      8 from collections import defaultdict

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend
```

```
File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\smallworld
 ↪py:301, in sigma(G, niter, nrand, seed)
    299 randMetrics = {"C": [], "L": []}
    300 for i in range(nrand):
--> 301     Gr = random_reference(G, niter=niter, seed=seed)
    302     randMetrics["C"].append(nx.transitivity(Gr))
    303     randMetrics["L"].append(nx.average_shortest_path_length(Gr))

File <class 'networkx.utils.decorators.argmap'> compilation 43:6, in␣
 ↪argmap_random_reference_37(G, niter, connectivity, seed, backend,␣
 ↪**backend_kwargs)
      4 import inspect
      5 import itertools
----> 6 import re
      7 import warnings
      8 from collections import defaultdict

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\smallworld
 ↪py:109, in random_reference(G, niter, connectivity, seed)
    106 G.remove_edge(c, d)
    108 # Check if the graph is still connected
--> 109 if connectivity and local_conn(G, a, b) == 0:
    110     # Not connected, revert the swap
    111     G.remove_edge(a, d)
    112     G.remove_edge(c, b)

File <class 'networkx.utils.decorators.argmap'> compilation 51:3, in␣
 ↪argmap_local_edge_connectivity_48(G, s, t, flow_func, auxiliary, residual,␣
 ↪cutoff, backend, **backend_kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools
```

```
File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\connectivit
 ↪py:643, in local_edge_connectivity(G, s, t, flow_func, auxiliary, residual,␣
 ↪cutoff)
    640 if flow_func is shortest_augmenting_path:
    641     kwargs["two_phase"] = True
--> 643 return nx.maximum_flow_value(H, s, t, **kwargs)

File <class 'networkx.utils.decorators.argmap'> compilation 59:3, in␣
 ↪argmap_maximum_flow_value_56(flowG, _s, _t, capacity, flow_func, backend,␣
 ↪**kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\flow\maxflo
 ↪py:302, in maximum_flow_value(flowG, _s, _t, capacity, flow_func, **kwargs)
    299 if not callable(flow_func):
    300     raise nx.NetworkXError("flow_func has to be callable.")
--> 302 R =␣
 ↪flow_func(flowG, _s, _t, capacity=capacity, value_only=True, **kwargs)
```

```
    304 return R.graph["flow_value"]

File <class 'networkx.utils.decorators.argmap'> compilation 63:3, in␣
↪argmap_edmonds_karp_60(G, s, t, capacity, residual, value_only, cutoff,␣
↪backend, **backend_kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi:
    971 # variable since "backend" is used in many comments and log/error␣
↪messages.
    972 backend_name = backend

File␣
↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith:s\flow\edmon
↪py:238, in edmonds_karp(G, s, t, capacity, residual, value_only, cutoff)
    120 @nx._dispatchable(edge_attrs={"capacity": float("inf")},␣
↪returns_graph=True)
    121 def edmonds_karp(
    122     G, s, t, capacity="capacity", residual=None, value_only=False,␣
↪cutoff=None
    123 ):
    124     """Find a maximum single-commodity flow using the Edmonds-Karp␣
↪algorithm.
    125
    126     This function returns the residual network resulting after computin
   (…)
    236
    237     """
--> 238     R = edmonds_karp_impl(G, s, t, capacity, residual, cutoff)
    239     R.graph["algorithm"] = "edmonds_karp"
    240     nx._clear_cache(R)

File␣
↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith:s\flow\edmon
↪py:104, in edmonds_karp_impl(G, s, t, capacity, residual, cutoff)
    101     raise nx.NetworkXError("source and sink are the same node")
    103 if residual is None:
--> 104     R = build_residual_network(G, capacity)
```

```
    105 else:
    106     R = residual

File <class 'networkx.utils.decorators.argmap'> compilation 67:3, in␣
 ↪argmap_build_residual_network_64(G, capacity, backend, **backend_kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\flow\utils
 ↪py:144, in build_residual_network(G, capacity)
    141             R.add_edge(v, u, capacity=0)
    142         else:
    143             # The edge (u, v) was added when (v, u) was visited.
--> 144             R[u][v]["capacity"] = r
    145 else:
    146     for u, v, attr in edge_list:
    147         # Add a pair of edges with equal residual capacities.

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\classes\ graph.
 ↪py:522, in Graph.__getitem__(self, n)
    498 def __getitem__(self, n):
    499     """Returns a dict of neighbors of node n.  Use: 'G[n]'.
    500
    501     Parameters
   (…)
    520     AtlasView({1: {}})
    521     """
--> 522     return self.adj[n]

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\classes\ :oreviews.
 ↪py:82, in AdjacencyView.__getitem__(self, name)
```

```
     81 def __getitem__(self, name):
---> 82     return AtlasView(self._atlas[name])

KeyboardInterrupt:
```

0.0.11 Comentario

Lo mismo pasa con este codigo

0.0.12 Ejercicio 8

Generar un modelo nulo aleatorio donde se tenga el mismo número de nodos y el mismo número de conexiones y comparar sus propiedades con el grafo original del cerebro

[45]:
```python
import networkx as nx
import numpy as np

# Crear el modelo nulo con el mismo número de nodos y conexiones
num_nodes = G.number_of_nodes()
num_edges = G.number_of_edges()
random_graph = nx.gnm_random_graph(num_nodes, num_edges)

# Comparar propiedades del grafo original y el modelo nulo
def calculate_graph_properties(graph):
    degree = np.mean([deg for _, deg in graph.degree()])
    clustering_coeff = nx.average_clustering(graph)
    small_world_coeff = nx.sigma(graph) if nx.is_connected(graph) else None
    closeness_centrality = np.mean(list(nx.closeness_centrality(graph).
 ↪values()))
    betweenness_centrality = np.mean(list(nx.betweenness_centrality(graph).
 ↪values()))

    return {
        "degree": degree,
        "clustering_coefficient": clustering_coeff,
        "small_world_coefficient": small_world_coeff,
        "closeness_centrality": closeness_centrality,
        "betweenness_centrality": betweenness_centrality,
    }

# Propiedades del grafo original
original_properties = calculate_graph_properties(G)
# Propiedades del modelo nulo
random_graph_properties = calculate_graph_properties(random_graph)

# Comparar resultados
print("Propiedades del grafo original:", original_properties)
```

```
print("Propiedades del modelo nulo:", random_graph_properties)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[45], line 26
     17     return {
     18         "degree": degree,
     19         "clustering_coefficient": clustering_coeff,
   (…)
     22         "betweenness_centrality": betweenness_centrality,
     23     }
     25 # Propiedades del grafo original
---> 26 original_properties = calculate_graph_properties(G)
     27 original_properties

Cell In[45], line 13, in calculate_graph_properties(graph)
     11 degree = np.mean([deg for _, deg in graph.degree()])
     12 clustering_coeff = nx.average_clustering(graph)
---> 13 small_world_coeff = nx.sigma(graph) if nx.is_connected(graph) else None
     14 closeness_centrality = np.mean(list(nx.closeness_centrality(graph).
 ↪values()))
     15 betweenness_centrality = np.mean(list(nx.betweenness_centrality(graph).
 ↪values()))

File <class 'networkx.utils.decorators.argmap'> compilation 36:6, in␣
 ↪argmap_sigma_30(G, niter, nrand, seed, backend, **backend_kwargs)
      4 import inspect
      5 import itertools
----> 6 import re
      7 import warnings
      8 from collections import defaultdict

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\smallworld
 ↪py:301, in sigma(G, niter, nrand, seed)
    299 randMetrics = {"C": [], "L": []}
```

```
    300 for i in range(nrand):
--> 301     Gr = random_reference(G, niter=niter, seed=seed)
    302     randMetrics["C"].append(nx.transitivity(Gr))
    303     randMetrics["L"].append(nx.average_shortest_path_length(Gr))

File <class 'networkx.utils.decorators.argmap'> compilation 43:6, in␣
 ↪argmap_random_reference_37(G, niter, connectivity, seed, backend,␣
 ↪**backend_kwargs)
      4 import inspect
      5 import itertools
----> 6 import re
      7 import warnings
      8 from collections import defaultdict

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\smallworld
 ↪py:109, in random_reference(G, niter, connectivity, seed)
    106 G.remove_edge(c, d)
    108 # Check if the graph is still connected
--> 109 if connectivity and local_conn(G, a, b) == 0:
    110     # Not connected, revert the swap
    111     G.remove_edge(a, d)
    112     G.remove_edge(c, b)

File <class 'networkx.utils.decorators.argmap'> compilation 51:3, in␣
 ↪argmap_local_edge_connectivity_48(G, s, t, flow_func, auxiliary, residual,␣
 ↪cutoff, backend, **backend_kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
```

```
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\connectivit
 ↪py:643, in local_edge_connectivity(G, s, t, flow_func, auxiliary, residual,␣
 ↪cutoff)
    640 if flow_func is shortest_augmenting_path:
    641     kwargs["two_phase"] = True
--> 643 return nx.maximum_flow_value(H, s, t, **kwargs)

File <class 'networkx.utils.decorators.argmap'> compilation 59:3, in␣
 ↪argmap_maximum_flow_value_56(flowG, _s, _t, capacity, flow_func, backend,␣
 ↪**kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\flow\maxfl
 ↪py:302, in maximum_flow_value(flowG, _s, _t, capacity, flow_func, **kwargs)
    299 if not callable(flow_func):
    300     raise nx.NetworkXError("flow_func has to be callable.")
--> 302 R =␣
 ↪flow_func(flowG, _s, _t, capacity=capacity, value_only=True, **kwargs)
    304 return R.graph["flow_value"]

File <class 'networkx.utils.decorators.argmap'> compilation 63:3, in␣
 ↪argmap_edmonds_karp_60(G, s, t, capacity, residual, value_only, cutoff,␣
 ↪backend, **backend_kwargs)
```

```
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\flow\edmon
 ↪py:238, in edmonds_karp(G, s, t, capacity, residual, value_only, cutoff)
    120 @nx._dispatchable(edge_attrs={"capacity": float("inf")},␣
 ↪returns_graph=True)
    121 def edmonds_karp(
    122     G, s, t, capacity="capacity", residual=None, value_only=False,␣
 ↪cutoff=None
    123 ):
    124     """Find a maximum single-commodity flow using the Edmonds-Karp␣
 ↪algorithm.
    125
    126     This function returns the residual network resulting after computin
   (…)
    236
    237     """
--> 238     R = edmonds_karp_impl(G, s, t, capacity, residual, cutoff)
    239     R.graph["algorithm"] = "edmonds_karp"
    240     nx._clear_cache(R)

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\flow\edmon
 ↪py:104, in edmonds_karp_impl(G, s, t, capacity, residual, cutoff)
    101     raise nx.NetworkXError("source and sink are the same node")
    103 if residual is None:
--> 104     R = build_residual_network(G, capacity)
    105 else:
    106     R = residual
```

```
File <class 'networkx.utils.decorators.argmap'> compilation 67:3, in␣
 ↪argmap_build_residual_network_64(G, capacity, backend, **backend_kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\flow\utils
 ↪py:137, in build_residual_network(G, capacity)
    135 for u, v, attr in edge_list:
    136     r = min(attr.get(capacity, inf), inf)
--> 137     if not R.has_edge(u, v):
    138         # Both (u, v) and (v, u) must be present in the residual
    139         # network.
    140         R.add_edge(u, v, capacity=r)
    141         R.add_edge(v, u, capacity=0)

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\classes\ graph.
 ↪py:1292, in Graph.has_edge(self, u, v)
   1289     else:
   1290         raise NetworkXError("update needs nodes or edges input")
-> 1292 def has_edge(self, u, v):
   1293     """Returns True if the edge (u, v) is in the graph.
   1294
   1295     This is the same as `v in G[u]` without KeyError exceptions.
   (…)
   1326
   1327     """
   1328     try:

KeyboardInterrupt:
```

0.0.13 Discusion

- En un modelo nulo aleatorio, el coeficiente de cluster suele ser bajo porque las conexiones entre nodos son distribuidas aleatoriamente y no tienden a formar triangulos ni comunidades densas
- Los grafos aleatorios tienden a tener una distancia promedio corta entre nodos. Esto se debe a que, aunque las coenxiones son aleatorias, la densidad de enlaces permite que la mayoria de los nodos esten conectados mediante pocas aristas
- Ademas, este modelo tiene una robustez estructural limitada y como el grafo origial es robusto, podria deberse a que contiene redundancias estructurales como conexiones a larga distancia o modulos que no estan presentes en el modelo nulo

0.0.14 Ejercicio 9

Generar un modelo nulo aleatorio donde se conserve la distribución de grado y comparar sus propiedades con el grafo original del cerebro.

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

# Grafo original
G_original = nx.from_numpy_array(coactivation_matrix)

# Generar un modelo nulo aleatorio con el mismo número de nodos y conexiones
G_random = nx.gnm_random_graph(G_original.number_of_nodes(), G_original.
 ↪number_of_edges())

# Calcular propiedades del grafo original
def calculate_graph_properties(G_original):
    degree = np.mean([deg for _, deg in G_original.degree()])
    clustering_coeff = nx.average_clustering(G_original)
    small_world_coeff = nx.sigma(G_original) if nx.is_connected(G_original)␣
 ↪else None
    closeness_centrality = np.mean(list(nx.closeness_centrality(G_original).
 ↪values()))
    betweenness_centrality = np.mean(list(nx.betweenness_centrality(G_original).
 ↪values()))
    return {
        'degree': degree,
        'clustering_coefficient': clustering_coeff,
        'small_world_coefficient': small_world_coeff,
        'closeness_centrality': closeness_centrality,
        'betweenness_centrality': betweenness_centrality
    }

# Propiedades del grafo original
original_properties = calculate_graph_properties(G_original)
# Propiedades del modelo nulo aleatorio
```

```
random_properties = calculate_graph_properties(G_random)

# Comparación de propiedades
print("Propiedades del grafo original:")
print(original_properties)

print("\nPropiedades del modelo nulo aleatorio:")
print(random_properties)
```

0.0.15 Ejercicio 10

Generar un modelo nulo utilizando una probabilidad de conexión en función de la distancia geométrica, con el mismo número de nodos y conexiones y compara sus propiedades y discutir la importancia de las conexiones a larga distancia en el cerebro.

[47]:
```
import networkx as nx
import numpy as np

# Grafo original
G_original = nx.from_numpy_array(coactivation_matrix)

# Generar un modelo nulo aleatorio preservando la distribución de grado
G_random_degree = nx.configuration_model([deg for _, deg in G_original.
 ↪degree()])

# Asegurarse de que no haya auto-lazos
G_random_degree = nx.Graph(G_random_degree)
G_random_degree.remove_edges_from(nx.selfloop_edges(G_random_degree))

# Calcular propiedades del grafo original
original_properties = calculate_graph_properties(G_original)
# Calcular propiedades del modelo nulo aleatorio preservando el grado
random_degree_properties = calculate_graph_properties(G_random_degree)

# Comparación de propiedades
print("Propiedades del grafo original:")
print(original_properties)

print("\nPropiedades del modelo nulo aleatorio (preservando distribución de␣
 ↪grados):")
print(random_degree_properties)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[47], line 15
     12 G_random_degree.remove_edges_from(nx.selfloop_edges(G_random_degree))
     14 # Calcular propiedades del grafo original
```

```
---> 15 original_properties = calculate_graph_properties(G_original)
     16 # Calcular propiedades del modelo nulo aleatorio preservando el grado
     17 random_degree_properties = calculate_graph_properties(G_random_degree)

Cell In[46], line 14, in calculate_graph_properties(graph)
     12 degree = np.mean([deg for _, deg in graph.degree()])
     13 clustering_coeff = nx.average_clustering(graph)
---> 14 small_world_coeff = nx.sigma(graph) if nx.is_connected(graph) else None
     15 closeness_centrality = np.mean(list(nx.closeness_centrality(graph).
 ↪values()))
     16 betweenness_centrality = np.mean(list(nx.betweenness_centrality(graph).
 ↪values()))

File <class 'networkx.utils.decorators.argmap'> compilation 36:6, in␣
 ↪argmap_sigma_30(G, niter, nrand, seed, backend, **backend_kwargs)
      4 import inspect
      5 import itertools
----> 6 import re
      7 import warnings
      8 from collections import defaultdict

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi:
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith:s\smallworld
 ↪py:301, in sigma(G, niter, nrand, seed)
    299 randMetrics = {"C": [], "L": []}
    300 for i in range(nrand):
--> 301     Gr = random_reference(G, niter=niter, seed=seed)
    302     randMetrics["C"].append(nx.transitivity(Gr))
    303     randMetrics["L"].append(nx.average_shortest_path_length(Gr))

File <class 'networkx.utils.decorators.argmap'> compilation 43:6, in␣
 ↪argmap_random_reference_37(G, niter, connectivity, seed, backend,␣
 ↪**backend_kwargs)
      4 import inspect
      5 import itertools
----> 6 import re
```

```
      7 import warnings
      8 from collections import defaultdict

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith:s\smallworld
 ↪py:109, in random_reference(G, niter, connectivity, seed)
    106 G.remove_edge(c, d)
    108 # Check if the graph is still connected
--> 109 if connectivity and local_conn(G, a, b) == 0:
    110     # Not connected, revert the swap
    111     G.remove_edge(a, d)
    112     G.remove_edge(c, b)

File <class 'networkx.utils.decorators.argmap'> compilation 51:3, in␣
 ↪argmap_local_edge_connectivity_48(G, s, t, flow_func, auxiliary, residual,␣
 ↪cutoff, backend, **backend_kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend
```

```
File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith ms\connectivit
 ↪py:643, in local_edge_connectivity(G, s, t, flow_func, auxiliary, residual,␣
 ↪cutoff)
    640 if flow_func is shortest_augmenting_path:
    641     kwargs["two_phase"] = True
--> 643 return nx.maximum_flow_value(H, s, t, **kwargs)

File <class 'networkx.utils.decorators.argmap'> compilation 59:3, in␣
 ↪argmap_maximum_flow_value_56(flowG, _s, _t, capacity, flow_func, backend,␣
 ↪**kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith ms\flow\maxflo
 ↪py:302, in maximum_flow_value(flowG, _s, _t, capacity, flow_func, **kwargs)
    299 if not callable(flow_func):
    300     raise nx.NetworkXError("flow_func has to be callable.")
--> 302 R =␣
 ↪flow_func(flowG, _s, _t, capacity=capacity, value_only=True, **kwargs)
    304 return R.graph["flow_value"]

File <class 'networkx.utils.decorators.argmap'> compilation 63:3, in␣
 ↪argmap_edmonds_karp_60(G, s, t, capacity, residual, value_only, cutoff,␣
 ↪backend, **backend_kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools
```

```
File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\flow\edmon
 ↪py:238, in edmonds_karp(G, s, t, capacity, residual, value_only, cutoff)
    120 @nx._dispatchable(edge_attrs={"capacity": float("inf")},␣
 ↪returns_graph=True)
    121 def edmonds_karp(
    122     G, s, t, capacity="capacity", residual=None, value_only=False,␣
 ↪cutoff=None
    123 ):
    124     """Find a maximum single-commodity flow using the Edmonds-Karp␣
 ↪algorithm.
    125
    126     This function returns the residual network resulting after computin
   (…)
    236
    237     """
--> 238     R = edmonds_karp_impl(G, s, t, capacity, residual, cutoff)
    239     R.graph["algorithm"] = "edmonds_karp"
    240     nx._clear_cache(R)

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\flow\edmon
 ↪py:104, in edmonds_karp_impl(G, s, t, capacity, residual, cutoff)
    101     raise nx.NetworkXError("source and sink are the same node")
    103 if residual is None:
--> 104     R = build_residual_network(G, capacity)
    105 else:
    106     R = residual

File <class 'networkx.utils.decorators.argmap'> compilation 67:3, in␣
 ↪argmap_build_residual_network_64(G, capacity, backend, **backend_kwargs)
      1 import bz2
      2 import collections
----> 3 import gzip
      4 import inspect
      5 import itertools
```

```
File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\utils\ba:kends.
 ↪py:967, in _dispatchable.__call__(self, backend, *args, **kwargs)
    965     if backend is not None and backend != "networkx":
    966         raise ImportError(f"'{backend}' backend is not installed")
--> 967     return self.orig_func(*args, **kwargs)
    969 # Use `backend_name` in this function instead of `backend`.
    970 # This is purely for aesthetics and to make it easier to search for thi
    971 # variable since "backend" is used in many comments and log/error␣
 ↪messages.
    972 backend_name = backend

File␣
 ↪~\AppData\Local\Programs\Python\Python312\Lib\site-packages\networkx\algorith s\flow\utils
 ↪py:111, in build_residual_network(G, capacity)
    108 inf = float("inf")
    109 # Extract edges with positive capacities. Self loops excluded.
    110 edge_list = [
--> 111     (u, v, attr)
    112     for u, v, attr in G.edges(data=True)
    113     if u != v and attr.get(capacity, inf) > 0
    114 ]
    115 # Simulate infinity with three times the sum of the finite edge␣
 ↪capacities
    116 # or any positive value if the sum is zero. This allows the
    117 # infinite-capacity edges to be distinguished for unboundedness detecti n
   (…)
    123 # than 1/3 of inf units of flow to t, there must be an infinite-capacit
    124 # s-t path in G.
    125 inf = (
    126     3
    127     * sum(
   (…)
    132     or 1
    133 )

KeyboardInterrupt:
```

0.0.16 Discusión

Las conexiones a larga distancia en el cerebro tienen una gran importancia para el funcionamiento general y la eficiencia de las redes neuronales. Estas conexiones permiten la comunicación entre diferentes regiones cerebrales, facilitando la integración de información de diversas partes del cerebro y permitiendo la realización de funciones cognitivas complejas

0.0.17 11

En esta clase aprendimos a aplicar herramientas matemáticas y computacionales para simular, analizar y comprender fenómenos complejos que ocurren en sistemas biológicos, incluso de analisis de datos mundiales. Estos estudios nos ayudan a predecir comportamientos y estudiar interacciones dentro de un sistema de manera más eficiente mediante la observación directa.

En particular, la teoría de grafos es una herramienta esencial en el estudio de la conectividad del cerebro, en donde los nodos pueden representar neuronas o regiones cerebrales, mientras que las aristas reflejan las conexiones sinápticas o la transferencia de información entre ellas. La teoría de grafos permite modelar y analizar estas complejas redes, lo que es fundamental para comprender cómo el cerebro organiza y transmite información.

Conocer herramientas de teoría de grafos es crucial para entender la conectividad cerebral porque permite identificar patrones, como la presencia de hubs, o la modularidad. Además, técnicas como la centralidad y el coeficiente de agrupamiento proporcionan información sobre la importancia relativa de las diferentes regiones cerebrales, ayudando a identificar qué áreas son claves para funciones cognitivas específicas.