

# proyecto-final-anelm

November 28, 2024

```
[ ]: # Proyecto final Modelos Computacionales
# Anel Mendiola
```

```
[133]: import bct
import scipy.io
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import networkx as nx
import pandas as pd
import networkx.algorithms.community as nx_comm
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

```
[134]: archivo = r"C:
    ↪\Users\ANEL\OneDrive\Escritorio\BCT\2019_03_03_BCT\data_and_demos\Coactivation_matrix.
    ↪mat"
```

```
[135]: mat_json = scipy.io.loadmat(archivo)
```

```
[136]: mat_dict = {k:v for k, v in mat_json.items() if k[0] != '_'}
mat_dict = {k:v for k, v in mat_json.items() if k[0] != '_'}
mat_dict
```

```
[136]: {'Coactivation_matrix': array([[0.          , 0.16071429, 0.11148649, ..., 0.
, 0.05045872,
0.1011236 ],
[0.16071429, 0.          , 0.06825939, ..., 0.          ,
0.06923077],
[0.11148649, 0.06825939, 0.          , ..., 0.03412969, 0.
, 0.          ],
...,
[0.          , 0.          , 0.03412969, ..., 0.          ,
0.          ],
[0.05045872, 0.          , 0.          , ..., 0.          ,
0.          ]])}
```

```

        0.09777778],
[0.1011236 , 0.06923077, 0.           , ..., 0.           , 0.09777778,
 0.          ]]),
'Coord': array([[ 7.24363636, 37.01090909,  9.42545455],
 [ 7.98653199, 46.22222222, 15.60942761],
 [ 7.55725191, 33.83206107, 23.51145038],
 ...,
 [-4.92385787, 15.31979695, 27.73604061],
 [-6.27312775, 34.70484581, -5.09251101],
 [-4.53874539, 46.53874539,  3.06273063]])}

```

[137]: mat\_dict.keys()

[137]: dict\_keys(['Coactivation\_matrix', 'Coord'])

```

[138]: # buscamos las dimensiones de la matriz
coactmat= mat_json['Coactivation_matrix']
coordenadas = mat_json['Coord']

coactmat_shape = coactmat.shape
coord_shape= coordenadas.shape

coactmat_shape, coord_shape

```

[138]: ((638, 638), (638, 3))

[10]: # 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 (incluidas en el archivo.mat).*

```

[141]: # Crear la función para generar grafos
def generate_graph(mat, umbral, coordenadas):
    # Matriz de adyacencia binaria
    matadj = (mat >= umbral).astype(int)
    np.fill_diagonal(matadj, 0)

    # Crear el grafo
    G = nx.from_numpy_array(matadj)

    # Añadir coordenadas 3D a los nodos
    for i, coord in enumerate(coordenadas):
        G.nodes[i]['pos'] = coord

    return G

# Generar los grafos con los umbrales 0.08, 0.09 y 0.10

```

```

# si le pongo el umbral de 0.8, 0.9 y 1, no se ven diferentes jejejeje, los
→puse más bajos

umbrales = [0.08, 0.09, 0.10]
graphs = {umbral: generate_graph(coactmat, umbral, coordenadas) for umbral in
         ↪umbrales}

# Función para graficar un grafo en 3D
def plot_graph_3d(G, title):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Obtener posiciones
    pos = nx.get_node_attributes(G, 'pos')
    if not pos:
        print("Error: Las posiciones 3D no están definidas en los nodos del"
              ↪grafo.")
    return

    x, y, z = np.array(list(pos.values())).T

    # Graficar nodos
    ax.scatter(x, y, z, c='darkgreen', s=50, label='Nodos')

    # Graficar aristas
    for edge in G.edges():
        x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
        ax.plot(x_edge, y_edge, z_edge, c='orange', alpha=0.5)

    ax.set_title(title)
    plt.legend()
    plt.show()

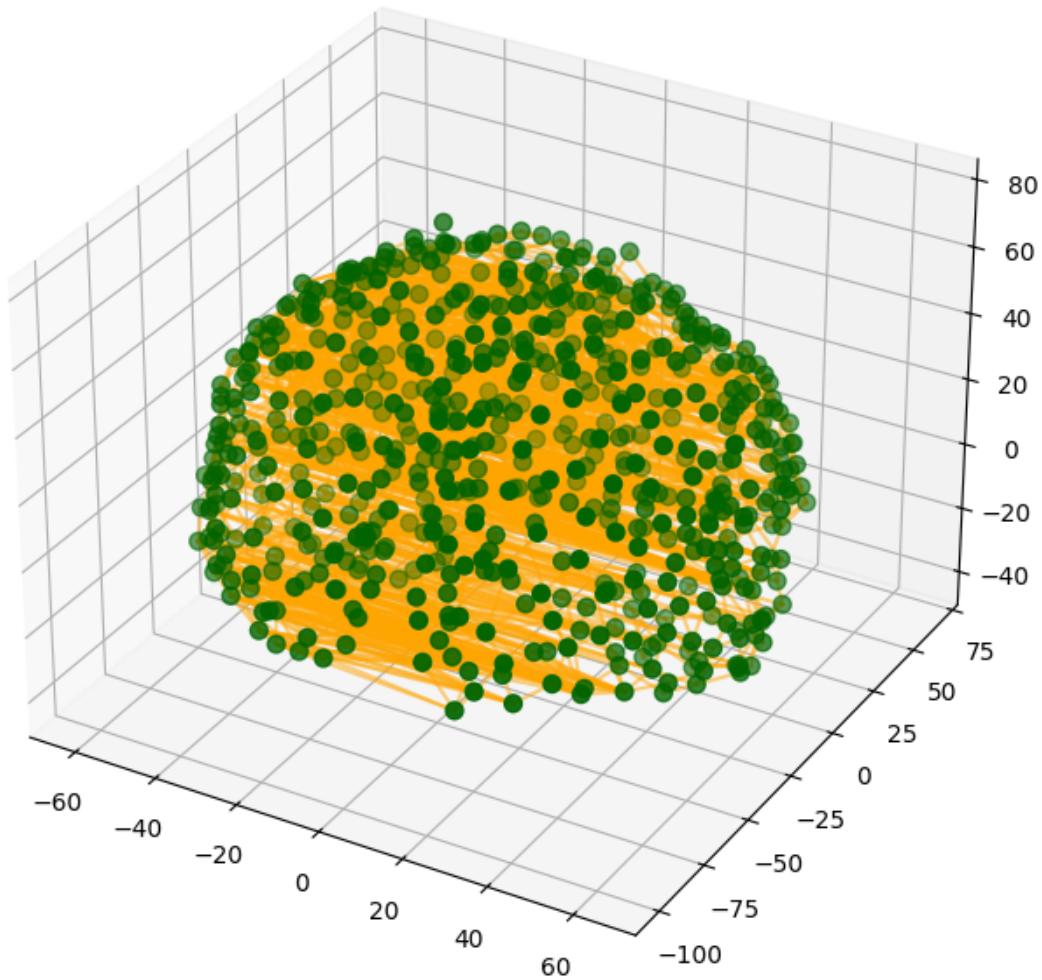
# Graficar los grafos generados
for umbral, graph in graphs.items():
    print(f"Grafo con umbral {umbral}: {graph.number_of_nodes()} nodos y {graph."
         ↪number_of_edges()} aristas.")
    plot_graph_3d(graph, f"Grafo con umbral {umbral}")

```

Grafo con umbral 0.08: 638 nodos y 2322 aristas.

Grafo con umbral 0.08

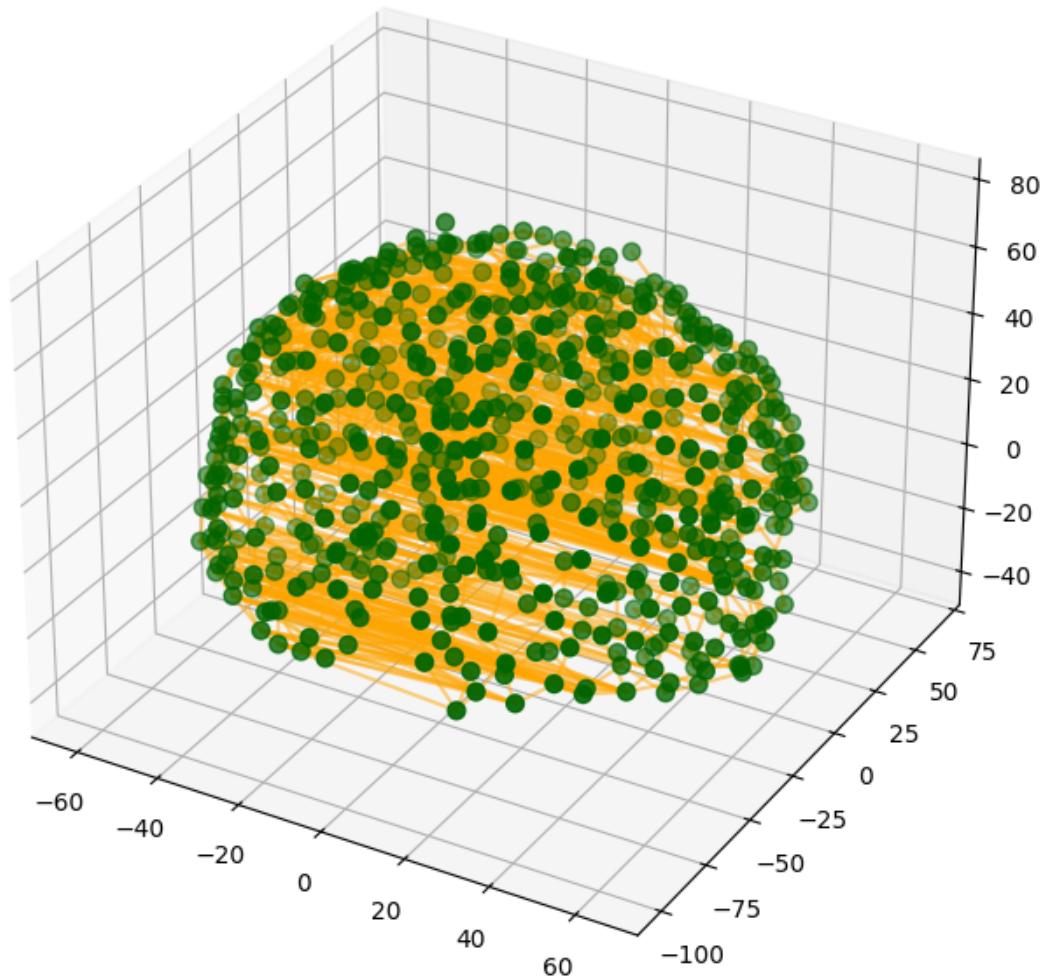
 Nodos



Grafo con umbral 0.09: 638 nodos y 1791 aristas.

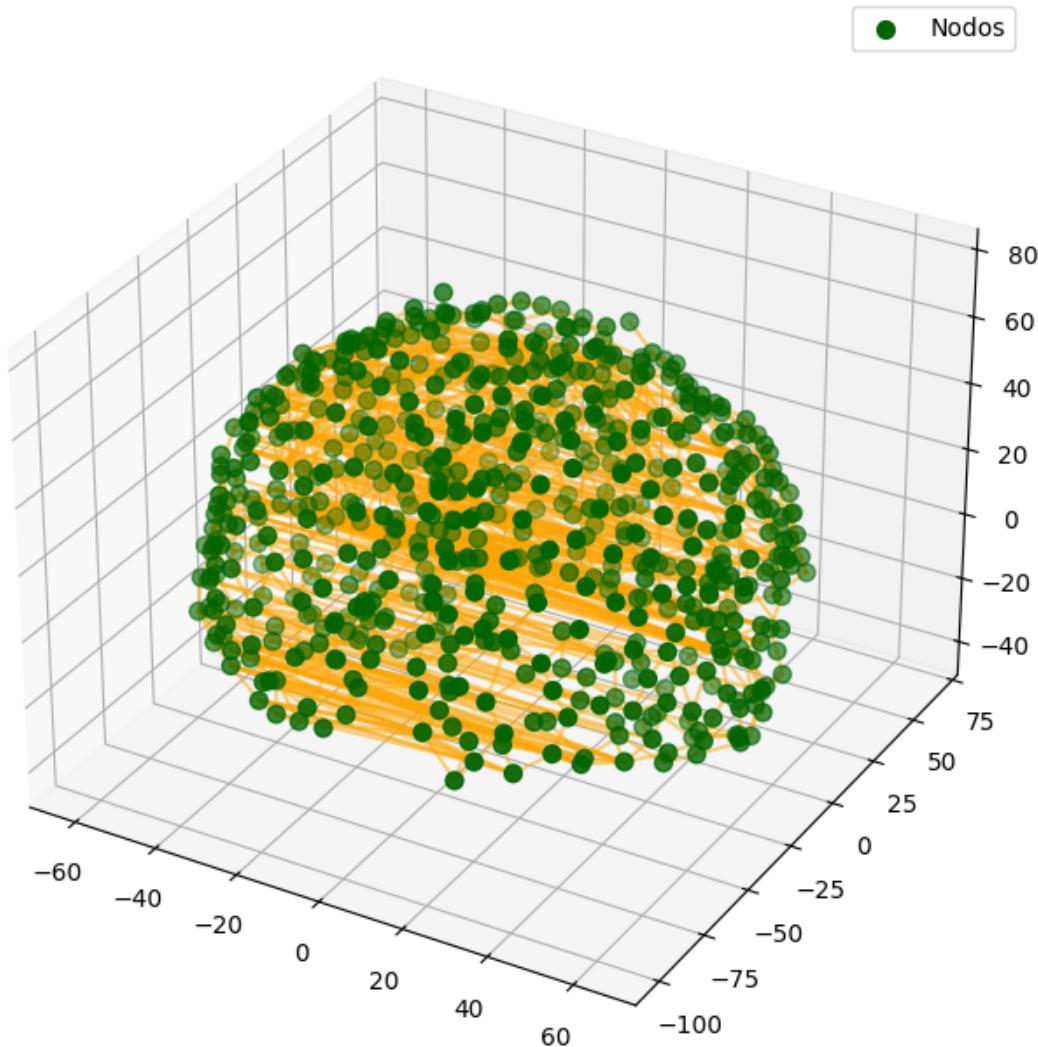
Grafo con umbral 0.09

 Nodos



Grafo con umbral 0.1: 638 nodos y 1445 aristas.

Grafo con umbral 0.1



```
[12]: # 2.  
# Con uno de los grafos en el punto uno con umbral 0.09, generar una animación  
# donde  
# se haga girar 360° el grafo del cerebro para visualizar las conexiones  
# establecidas
```

```
[147]: from matplotlib.animation import FuncAnimation  
  
def animacion_grafo(G, ej2):  
    fig = plt.figure()  
    ax= fig.add_subplot(111, projection ='3d')  
    pos= nx.get_node_attributes(G, 'pos')
```

```

x, y, z= np.array(list(pos.values())).T

#nodos y aristas
scatter= ax.scatter(x,y,z, c='darksalmon', s=20, label='Nodos')
lines = []
for edge in G.edges():
    x_edge, y_edge, z_edge = zip(pos[edge[0]],pos[edge[1]])
    lines.append(ax.plot( x_edge, y_edge, z_edge, c= 'lightseagreen', alpha= 0.5)[0])

ax.set_title("Grafo con umbral 0.09")

# rotación
def update(frame):
    ax.view_init(elev=10, azim=frame)
    return scatter, *lines

# animación
ani = FuncAnimation(fig, update, frames= np.arange(0, 360, 2), interval=50, blit= False)

#guardamos la animación como gif para que rote todo el tiempo

ani.save(ej2, writer='pillow', fps=20)
plt.close(fig)
# en otros ejercicios usé otra animación, pero no sale rotando todo el tiempo
# con un gif se ve más bonito

ruta_archivo = r"C:\Users\ANEL\grafo1.gif"
grafo09= graphs[0.09]
animacion_grafo(grafo09, ruta_archivo)

```

[148]: # a ver si salió la animación :o  
from IPython.display import Image  
Image(r"C:\Users\ANEL\grafo1.gif")

[148]: <IPython.core.display.Image object>

[15]: #3.  
# Encontrar los hubs del grafo, y establecer el tamaño del nodo proporcional al valor del grado

[155]: def plot\_hubs(G, ej3a):  
# Obtener posiciones 3D de los nodos  
pos = nx.get\_node\_attributes(G, 'pos')  
x, y, z = np.array(list(pos.values())).T

```

# Calcular el grado de los nodos
grados = dict(G.degree())
print("Grados de los nodos:", grados)

# Escalar el tamaño de los nodos proporcional al grado
tam_min, tam_max = 0.01, 700
grad_max = max(grados.values())
tamnod = [
    tam_min + (grado / grad_max) * (tam_max - tam_min) for grado in grados.
    values()
]

# Identificar hubs
umbral = np.percentile(list(grados.values()), 90)
hub_nodes = [node for node, grado in grados.items() if grado >= umbral]
print(f"Nodos hubs (grado {umbral}): {hub_nodes}")

# Colorear los hubs de rojo y los demás nodos de naranja
nodcol = ['orangered' if node in hub_nodes else 'orange' for node in G.
nodes()]

# Crear figura 3D
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Graficar nodos
scatter = ax.scatter(x, y, z, c=nodcol, s=tamnod, label='Nodos')

# Graficar aristas
for edge in G.edges():
    x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
    ax.plot(x_edge, y_edge, z_edge, c='seagreen', alpha=0.5, linewidth=0.5)

# Ajustar título y leyenda
ax.set_title(ej3a)
ax.legend(["Hubs en rojo y otros nodos en naranja"])
plt.show()

# Grafo con umbral de 0.09
grafo09 = generate_graph(coactmat, 0.09, coordenadas)
plot_hubs(grafo01, "Grafo con umbral 0.09 mostrando los hubs")

```

Grados de los nodos: {0: 5, 1: 5, 2: 4, 3: 3, 4: 3, 5: 3, 6: 4, 7: 8, 8: 3, 9: 3, 10: 2, 11: 3, 12: 3, 13: 1, 14: 1, 15: 4, 16: 5, 17: 3, 18: 4, 19: 8, 20: 3, 21: 3, 22: 4, 23: 3, 24: 4, 25: 3, 26: 2, 27: 4, 28: 4, 29: 4, 30: 3, 31: 5, 32: 4, 33: 3, 34: 4, 35: 5, 36: 4, 37: 6, 38: 16, 39: 3, 40: 5, 41: 6, 42: 5, 43: 3,

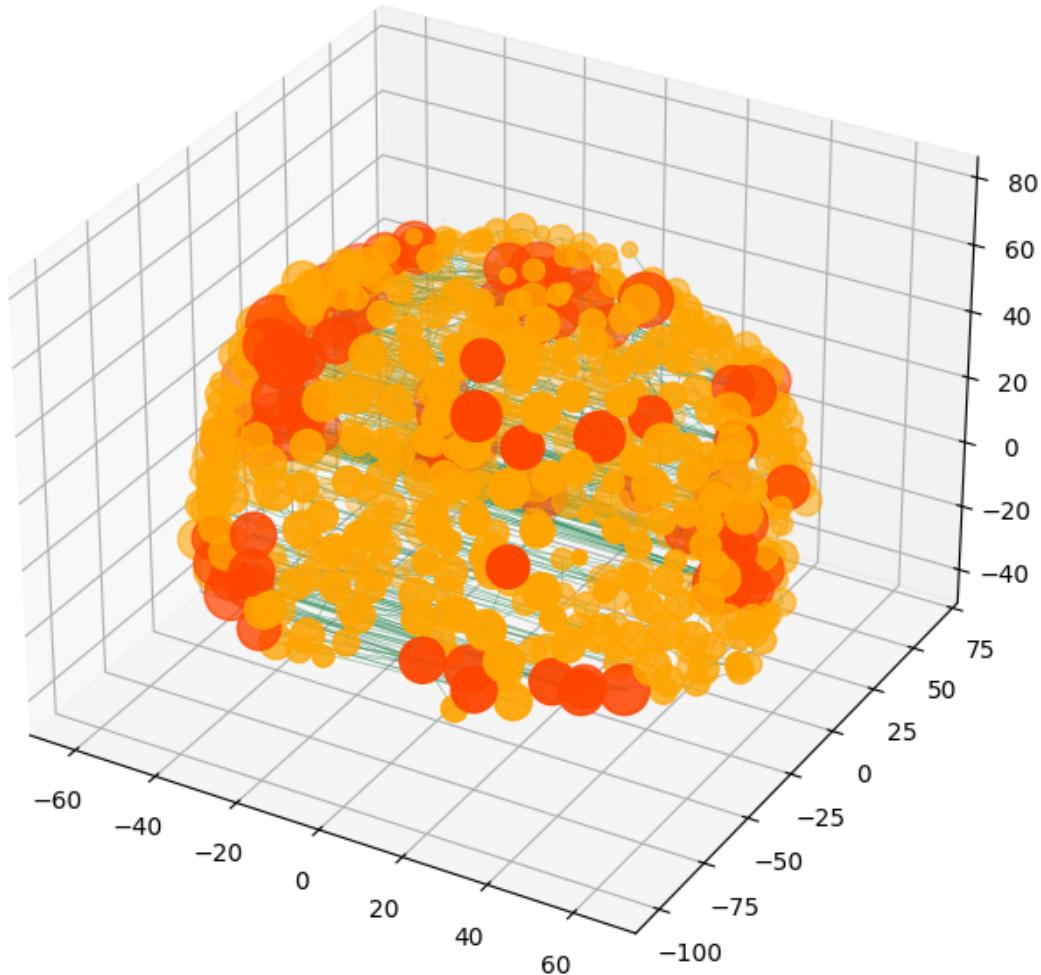
44: 6, 45: 4, 46: 3, 47: 2, 48: 4, 49: 4, 50: 4, 51: 3, 52: 2, 53: 3, 54: 5, 55: 4, 56: 2, 57: 5, 58: 2, 59: 4, 60: 2, 61: 3, 62: 5, 63: 5, 64: 6, 65: 6, 66: 4, 67: 5, 68: 5, 69: 7, 70: 10, 71: 4, 72: 4, 73: 5, 74: 4, 75: 3, 76: 4, 77: 4, 78: 3, 79: 2, 80: 4, 81: 6, 82: 3, 83: 4, 84: 5, 85: 3, 86: 5, 87: 5, 88: 3, 89: 3, 90: 2, 91: 3, 92: 3, 93: 5, 94: 2, 95: 3, 96: 3, 97: 4, 98: 5, 99: 4, 100: 9, 101: 3, 102: 3, 103: 3, 104: 5, 105: 3, 106: 4, 107: 3, 108: 4, 109: 3, 110: 5, 111: 3, 112: 4, 113: 5, 114: 2, 115: 3, 116: 4, 117: 3, 118: 4, 119: 3, 120: 5, 121: 8, 122: 5, 123: 3, 124: 4, 125: 5, 126: 7, 127: 3, 128: 5, 129: 2, 130: 6, 131: 6, 132: 4, 133: 3, 134: 5, 135: 9, 136: 4, 137: 3, 138: 4, 139: 0, 140: 3, 141: 2, 142: 3, 143: 2, 144: 4, 145: 2, 146: 2, 147: 2, 148: 5, 149: 5, 150: 3, 151: 3, 152: 1, 153: 7, 154: 7, 155: 2, 156: 3, 157: 3, 158: 6, 159: 3, 160: 4, 161: 2, 162: 3, 163: 2, 164: 1, 165: 5, 166: 2, 167: 4, 168: 1, 169: 2, 170: 1, 171: 2, 172: 2, 173: 4, 174: 3, 175: 1, 176: 2, 177: 1, 178: 2, 179: 1, 180: 2, 181: 2, 182: 3, 183: 3, 184: 2, 185: 0, 186: 13, 187: 3, 188: 3, 189: 3, 190: 2, 191: 3, 192: 3, 193: 6, 194: 6, 195: 4, 196: 6, 197: 5, 198: 2, 199: 3, 200: 2, 201: 3, 202: 9, 203: 3, 204: 4, 205: 1, 206: 4, 207: 2, 208: 6, 209: 1, 210: 3, 211: 8, 212: 4, 213: 8, 214: 8, 215: 1, 216: 0, 217: 6, 218: 4, 219: 2, 220: 5, 221: 2, 222: 3, 223: 5, 224: 2, 225: 3, 226: 7, 227: 2, 228: 5, 229: 2, 230: 14, 231: 4, 232: 7, 233: 4, 234: 1, 235: 15, 236: 2, 237: 8, 238: 4, 239: 2, 240: 5, 241: 2, 242: 5, 243: 3, 244: 5, 245: 5, 246: 5, 247: 3, 248: 3, 249: 1, 250: 5, 251: 4, 252: 2, 253: 10, 254: 3, 255: 1, 256: 2, 257: 3, 258: 4, 259: 7, 260: 8, 261: 9, 262: 12, 263: 9, 264: 9, 265: 6, 266: 9, 267: 9, 268: 9, 269: 5, 270: 5, 271: 8, 272: 6, 273: 5, 274: 6, 275: 6, 276: 4, 277: 4, 278: 5, 279: 8, 280: 6, 281: 7, 282: 5, 283: 3, 284: 8, 285: 7, 286: 5, 287: 1, 288: 4, 289: 6, 290: 4, 291: 5, 292: 7, 293: 2, 294: 1, 295: 3, 296: 5, 297: 3, 298: 5, 299: 3, 300: 0, 301: 3, 302: 4, 303: 7, 304: 5, 305: 3, 306: 2, 307: 1, 308: 9, 309: 7, 310: 1, 311: 3, 312: 4, 313: 3, 314: 3, 315: 3, 316: 1, 317: 5, 318: 3, 319: 2, 320: 3, 321: 4, 322: 3, 323: 4, 324: 2, 325: 1, 326: 3, 327: 7, 328: 12, 329: 5, 330: 15, 331: 7, 332: 3, 333: 3, 334: 13, 335: 4, 336: 3, 337: 5, 338: 4, 339: 6, 340: 2, 341: 5, 342: 4, 343: 4, 344: 7, 345: 8, 346: 11, 347: 4, 348: 6, 349: 3, 350: 8, 351: 9, 352: 7, 353: 9, 354: 5, 355: 2, 356: 15, 357: 8, 358: 7, 359: 5, 360: 4, 361: 8, 362: 11, 363: 4, 364: 4, 365: 7, 366: 5, 367: 8, 368: 7, 369: 7, 370: 5, 371: 7, 372: 6, 373: 9, 374: 5, 375: 7, 376: 7, 377: 4, 378: 4, 379: 6, 380: 4, 381: 4, 382: 5, 383: 5, 384: 2, 385: 5, 386: 3, 387: 3, 388: 5, 389: 2, 390: 3, 391: 1, 392: 8, 393: 8, 394: 3, 395: 3, 396: 2, 397: 10, 398: 6, 399: 5, 400: 18, 401: 7, 402: 1, 403: 9, 404: 7, 405: 4, 406: 6, 407: 11, 408: 4, 409: 3, 410: 9, 411: 3, 412: 6, 413: 0, 414: 7, 415: 4, 416: 11, 417: 3, 418: 8, 419: 3, 420: 4, 421: 4, 422: 2, 423: 4, 424: 1, 425: 4, 426: 3, 427: 5, 428: 4, 429: 3, 430: 6, 431: 5, 432: 4, 433: 4, 434: 5, 435: 3, 436: 6, 437: 3, 438: 2, 439: 5, 440: 3, 441: 4, 442: 4, 443: 2, 444: 5, 445: 2, 446: 3, 447: 4, 448: 3, 449: 2, 450: 5, 451: 3, 452: 8, 453: 8, 454: 10, 455: 7, 456: 5, 457: 9, 458: 5, 459: 5, 460: 4, 461: 6, 462: 2, 463: 4, 464: 4, 465: 8, 466: 5, 467: 2, 468: 2, 469: 2, 470: 5, 471: 3, 472: 6, 473: 10, 474: 5, 475: 6, 476: 3, 477: 10, 478: 6, 479: 5, 480: 3, 481: 11, 482: 16, 483: 4, 484: 1, 485: 16, 486: 2, 487: 3, 488: 10, 489: 3, 490: 2, 491: 13, 492: 2, 493: 3, 494: 10, 495: 4, 496: 8, 497: 2, 498: 2, 499: 2, 500: 8, 501: 4, 502: 5, 503: 4, 504: 3, 505: 6, 506: 5, 507: 4, 508: 3, 509: 4, 510: 4, 511: 3, 512: 2, 513: 7, 514: 4, 515: 2, 516: 5, 517: 4, 518: 2, 519: 2, 520: 2, 521: 3, 522: 3, 523: 2, 524: 1, 525: 1, 526: 5, 527: 1,

528: 6, 529: 3, 530: 1, 531: 2, 532: 11, 533: 3, 534: 3, 535: 2, 536: 6, 537: 3, 538: 4, 539: 4, 540: 2, 541: 3, 542: 4, 543: 3, 544: 4, 545: 3, 546: 5, 547: 3, 548: 4, 549: 7, 550: 3, 551: 6, 552: 9, 553: 9, 554: 3, 555: 4, 556: 4, 557: 4, 558: 4, 559: 4, 560: 5, 561: 3, 562: 4, 563: 5, 564: 5, 565: 4, 566: 4, 567: 4, 568: 5, 569: 2, 570: 6, 571: 4, 572: 4, 573: 3, 574: 3, 575: 4, 576: 5, 577: 5, 578: 3, 579: 3, 580: 6, 581: 3, 582: 4, 583: 0, 584: 4, 585: 2, 586: 1, 587: 3, 588: 2, 589: 4, 590: 2, 591: 3, 592: 1, 593: 1, 594: 1, 595: 2, 596: 2, 597: 4, 598: 5, 599: 11, 600: 6, 601: 9, 602: 5, 603: 15, 604: 5, 605: 5, 606: 11, 607: 10, 608: 5, 609: 2, 610: 3, 611: 7, 612: 8, 613: 9, 614: 4, 615: 9, 616: 10, 617: 10, 618: 8, 619: 15, 620: 2, 621: 7, 622: 8, 623: 6, 624: 2, 625: 3, 626: 5, 627: 6, 628: 6, 629: 10, 630: 5, 631: 3, 632: 4, 633: 3, 634: 3, 635: 3, 636: 3, 637: 4}

Nodos hubs (grado 8.0): [7, 19, 38, 70, 100, 121, 135, 186, 202, 211, 213, 214, 230, 235, 237, 253, 260, 261, 262, 263, 264, 266, 267, 268, 271, 279, 284, 308, 328, 330, 334, 345, 346, 350, 351, 353, 356, 357, 361, 362, 367, 373, 392, 393, 397, 400, 403, 407, 410, 416, 418, 452, 453, 454, 457, 465, 473, 477, 481, 482, 485, 488, 491, 494, 496, 500, 532, 552, 553, 599, 601, 603, 606, 607, 612, 613, 615, 616, 617, 618, 619, 622, 629]

Grafo con umbral 0.09 mostrando los hubs

Hubs en rojo y otros nodos en naranja



[ ]: #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.

```
[164]: import community as community_louvain  
import seaborn as sns  
import matplotlib.pyplot as plt  
import numpy as np  
import networkx as nx
```

```
[169]: def robust_corrcoef(mat):

    std_dev = np.std(mat, axis=1)
    valid_rows = std_dev > 0 # aquí se filtran las filas que no son constantes
    if np.sum(valid_rows) == 0:
        raise ValueError("Todas las filas tienen desviación estándar cero.")
    filtered_mat = mat[valid_rows] [:, valid_rows]
    return np.corrcoef(filtered_mat)

def modulos(G, ej4):

    # Sacamos la matriz de adyacencia
    matadj = nx.to_numpy_array(G)
    print("Ya se calculó la matriz de adyacencia")

    # Calcular la matriz de emparejamiento
    try:
        corrrmat = robust_corrcoef(matadj)
    except ValueError as e:
        print(e)
        return # Salir si no hay filas válidas

    print("Ya se calculó la matriz de emparejamiento")

    # Visualizar la matriz de emparejamiento
    sns.heatmap(corrrmat, cmap="coolwarm", center=0)
    plt.title("Matriz de emparejamiento (correlación)")
    plt.show()

    # Partición por módulos usando el algoritmo de Louvain
    print("Calculando la partición de los nodos en módulos")
    particion = community_louvain.best_partition(G)
    print("Ya quedó la partición:", particion)

    # Colorear nodos según su módulo
    nodcol = list(particion.values())

    # Cálculo de la modularidad
    modularidad = community_louvain.modularity(particion, G)
    print(f"Modularidad calculada: {modularidad:.4f}")

    # Figura en 3D
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Obtener posiciones de los nodos
    pos = nx.get_node_attributes(G, 'pos')
```

```

if not pos:
    raise ValueError("El grafo no tiene posiciones 3D asignadas a los nodos.
")
x, y, z = np.array(list(pos.values())).T

# Graficar los nodos
scatter = ax.scatter(x, y, z, c=nodcol, cmap="viridis", s=50, label='Nodos')

# Graficar las aristas
for edge in G.edges():
    x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
    ax.plot(x_edge, y_edge, z_edge, c='b', alpha=0.5, linewidth=0.5)

# Título y leyenda
ax.set_title(f"ej4\nModularidad: {modularidad:.4f}")
plt.colorbar(scatter, ax=ax, label="Módulo")
plt.show()

# Grafo con umbral 0.1
grafo09 = generate_graph(coactmat, 0.09, coordenadas)

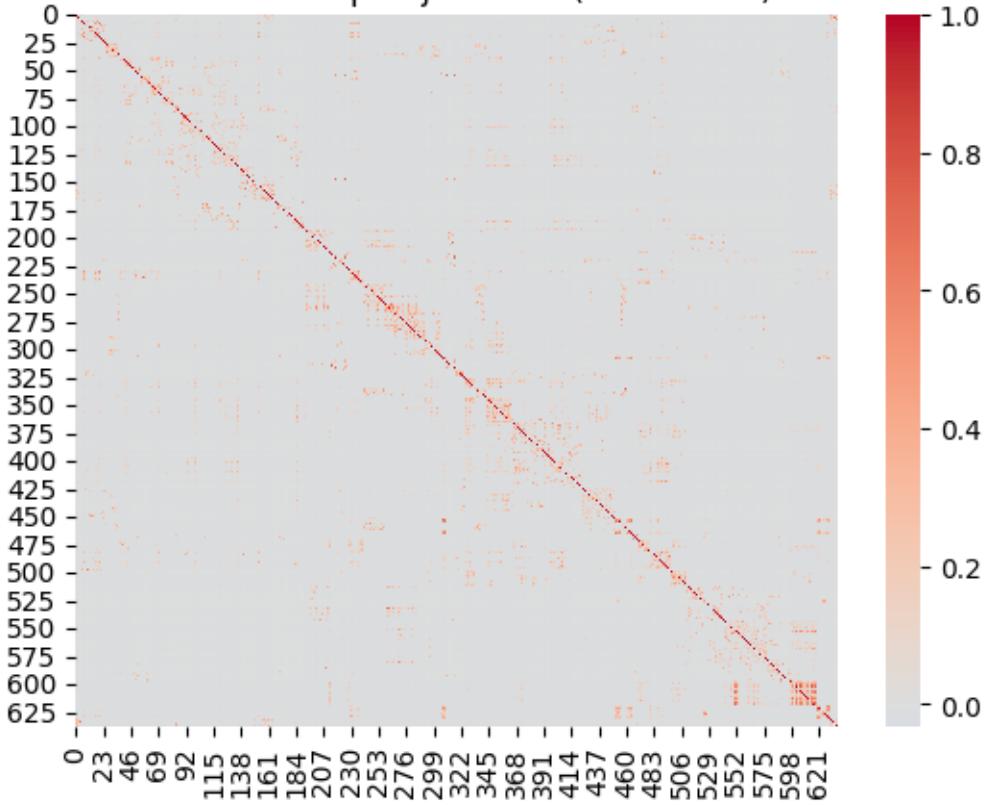
# Partición por módulos y graficar
modulos(grafo09, "Partición por módulos del grafo con umbral 0.09")

```

Ya se calculó la matriz de adyacencia

Ya se calculó la matriz de emparejamiento

Matriz de emparejamiento (correlación)



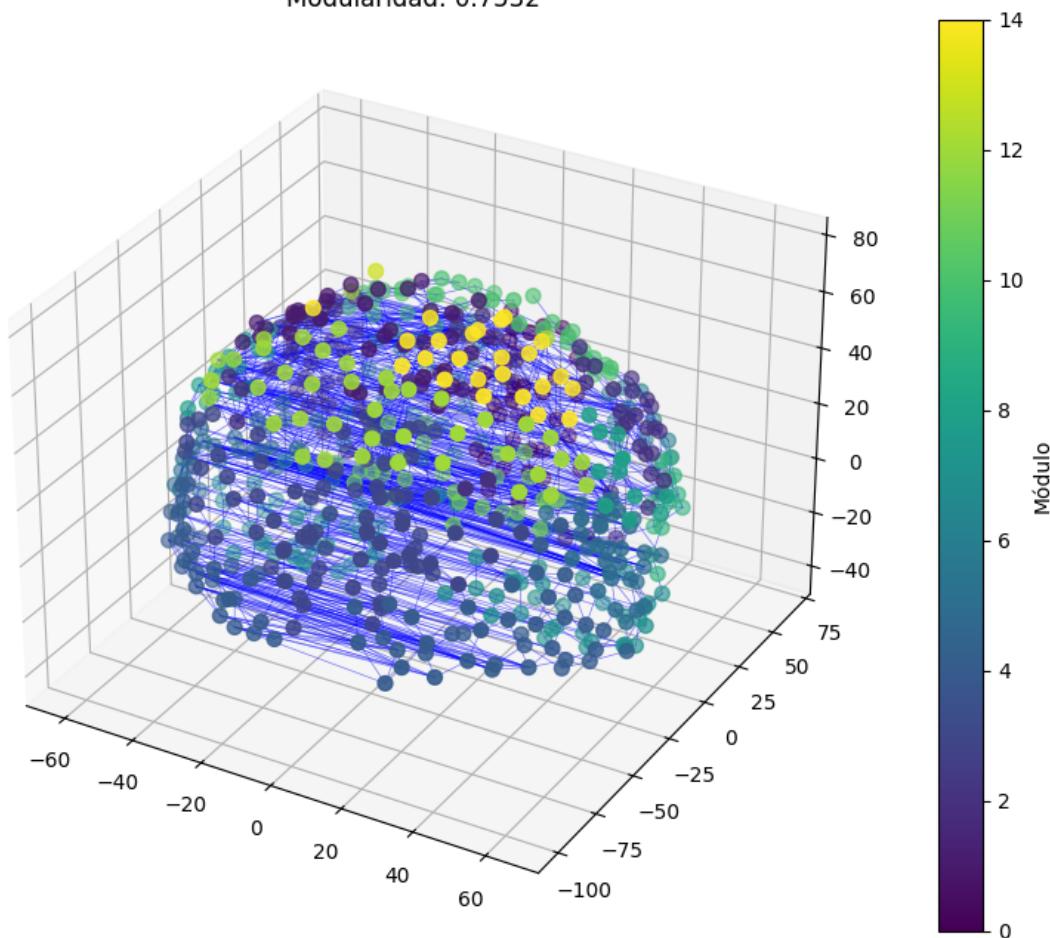
Calculando la partición de los nodos en módulos

Ya quedó la partición: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 1, 6: 1, 7: 2, 8: 1, 9: 3, 10: 1, 11: 3, 12: 1, 13: 3, 14: 3, 15: 1, 16: 2, 17: 1, 18: 1, 19: 2, 20: 1, 21: 3, 22: 3, 23: 3, 24: 3, 25: 3, 26: 3, 27: 3, 28: 3, 29: 3, 30: 3, 31: 3, 32: 3, 33: 3, 34: 3, 35: 4, 36: 3, 37: 2, 38: 2, 39: 2, 40: 2, 41: 2, 42: 2, 43: 2, 44: 2, 45: 2, 46: 5, 47: 6, 48: 6, 49: 6, 50: 2, 51: 6, 52: 7, 53: 6, 54: 7, 55: 9, 56: 2, 57: 2, 58: 2, 59: 9, 60: 9, 61: 9, 62: 2, 63: 2, 64: 2, 65: 2, 66: 6, 67: 2, 68: 2, 69: 2, 70: 2, 71: 6, 72: 9, 73: 2, 74: 9, 75: 9, 76: 2, 77: 9, 78: 9, 79: 9, 80: 2, 81: 0, 82: 0, 83: 0, 84: 0, 85: 0, 86: 0, 87: 6, 88: 10, 89: 2, 90: 10, 91: 6, 92: 10, 93: 2, 94: 2, 95: 10, 96: 10, 97: 2, 98: 6, 99: 10, 100: 2, 101: 10, 102: 10, 103: 6, 104: 6, 105: 10, 106: 6, 107: 6, 108: 6, 109: 6, 110: 6, 111: 6, 112: 6, 113: 6, 114: 10, 115: 6, 116: 2, 117: 10, 118: 2, 119: 6, 120: 6, 121: 2, 122: 6, 123: 10, 124: 2, 125: 2, 126: 2, 127: 6, 128: 6, 129: 2, 130: 2, 131: 2, 132: 10, 133: 2, 134: 10, 135: 12, 136: 6, 137: 6, 138: 10, 139: 13, 140: 10, 141: 10, 142: 6, 143: 10, 144: 10, 145: 10, 146: 6, 147: 6, 148: 7, 149: 0, 150: 0, 151: 10, 152: 10, 153: 0, 154: 2, 155: 0, 156: 10, 157: 0, 158: 0, 159: 0, 160: 0, 161: 0, 162: 0, 163: 0, 164: 0, 165: 0, 166: 0, 167: 2, 168: 10, 169: 0, 170: 6, 171: 7, 172: 6, 173: 6, 174: 6, 175: 6, 176: 0, 177: 6, 178: 6, 179: 6, 180: 10, 181: 6, 182: 10, 183: 10, 184: 10, 185: 14, 186: 12, 187: 10, 188: 10, 189: 10, 190: 10, 191: 10, 192: 10, 193: 14, 194: 4, 195: 4, 196: 7, 197: 4, 198: 7, 199: 7, 200: 7, 201: 7, 202: 4, 203: 4, 204: 7, 205: 7,}

206: 4, 207: 4, 208: 4, 209: 7, 210: 7, 211: 4, 212: 4, 213: 5, 214: 5, 215: 7, 216: 7, 217: 7, 218: 7, 219: 3, 220: 3, 221: 3, 222: 3, 223: 3, 224: 7, 225: 7, 226: 7, 227: 7, 228: 5, 229: 5, 230: 2, 231: 2, 232: 2, 233: 5, 234: 2, 235: 2, 236: 2, 237: 2, 238: 5, 239: 5, 240: 5, 241: 5, 242: 3, 243: 4, 244: 3, 245: 4, 246: 3, 247: 3, 248: 4, 249: 3, 250: 3, 251: 3, 252: 4, 253: 4, 254: 4, 255: 3, 256: 3, 257: 7, 258: 3, 259: 4, 260: 4, 261: 4, 262: 4, 263: 4, 264: 4, 265: 4, 266: 4, 267: 4, 268: 3, 269: 4, 270: 4, 271: 4, 272: 4, 273: 4, 274: 4, 275: 3, 276: 3, 277: 3, 278: 3, 279: 4, 280: 3, 281: 4, 282: 4, 283: 4, 284: 3, 285: 4, 286: 3, 287: 3, 288: 4, 289: 3, 290: 3, 291: 3, 292: 3, 293: 12, 294: 3, 295: 3, 296: 2, 297: 4, 298: 2, 299: 4, 300: 3, 301: 3, 302: 3, 303: 3, 304: 4, 305: 3, 306: 7, 307: 11, 308: 11, 309: 11, 310: 7, 311: 7, 312: 7, 313: 7, 314: 7, 315: 7, 316: 7, 317: 7, 318: 7, 319: 1, 320: 1, 321: 1, 322: 1, 323: 1, 324: 1, 325: 14, 326: 14, 327: 12, 328: 2, 329: 8, 330: 12, 331: 12, 332: 12, 333: 12, 334: 2, 335: 3, 336: 3, 337: 3, 338: 3, 339: 3, 340: 4, 341: 4, 342: 3, 343: 3, 344: 12, 345: 12, 346: 12, 347: 12, 348: 12, 349: 12, 350: 12, 351: 12, 352: 12, 353: 3, 354: 12, 355: 12, 356: 2, 357: 12, 358: 12, 359: 12, 360: 12, 361: 12, 362: 12, 363: 14, 364: 12, 365: 12, 366: 14, 367: 8, 368: 1, 369: 1, 370: 8, 371: 8, 372: 1, 373: 8, 374: 8, 375: 1, 376: 8, 377: 8, 378: 1, 379: 8, 380: 1, 381: 1, 382: 14, 383: 8, 384: 8, 385: 8, 386: 14, 387: 14, 388: 14, 389: 14, 390: 14, 391: 14, 392: 8, 393: 8, 394: 8, 395: 14, 396: 14, 397: 2, 398: 2, 399: 8, 400: 2, 401: 2, 402: 1, 403: 2, 404: 1, 405: 2, 406: 1, 407: 2, 408: 1, 409: 1, 410: 1, 411: 2, 412: 14, 413: 14, 414: 8, 415: 14, 416: 2, 417: 14, 418: 2, 419: 14, 420: 8, 421: 2, 422: 14, 423: 14, 424: 14, 425: 12, 426: 12, 427: 3, 428: 3, 429: 12, 430: 12, 431: 3, 432: 12, 433: 12, 434: 12, 435: 1, 436: 12, 437: 12, 438: 3, 439: 12, 440: 12, 441: 14, 442: 3, 443: 3, 444: 12, 445: 3, 446: 3, 447: 12, 448: 3, 449: 3, 450: 12, 451: 3, 452: 11, 453: 11, 454: 11, 455: 11, 456: 3, 457: 4, 458: 3, 459: 3, 460: 3, 461: 3, 462: 11, 463: 11, 464: 11, 465: 11, 466: 0, 467: 0, 468: 7, 469: 6, 470: 0, 471: 0, 472: 5, 473: 5, 474: 5, 475: 5, 476: 5, 477: 5, 478: 5, 479: 5, 480: 5, 481: 1, 482: 2, 483: 10, 484: 10, 485: 2, 486: 1, 487: 1, 488: 1, 489: 1, 490: 1, 491: 2, 492: 10, 493: 10, 494: 2, 495: 1, 496: 1, 497: 1, 498: 1, 499: 8, 500: 8, 501: 5, 502: 8, 503: 8, 504: 5, 505: 8, 506: 8, 507: 8, 508: 5, 509: 12, 510: 8, 511: 12, 512: 7, 513: 4, 514: 4, 515: 4, 516: 7, 517: 7, 518: 7, 519: 7, 520: 4, 521: 4, 522: 7, 523: 7, 524: 7, 525: 11, 526: 11, 527: 11, 528: 11, 529: 7, 530: 7, 531: 7, 532: 4, 533: 7, 534: 4, 535: 7, 536: 4, 537: 4, 538: 4, 539: 4, 540: 7, 541: 4, 542: 7, 543: 5, 544: 5, 545: 4, 546: 7, 547: 7, 548: 7, 549: 5, 550: 5, 551: 4, 552: 5, 553: 5, 554: 5, 555: 5, 556: 4, 557: 4, 558: 5, 559: 7, 560: 4, 561: 7, 562: 5, 563: 5, 564: 5, 565: 7, 566: 4, 567: 4, 568: 5, 569: 4, 570: 4, 571: 5, 572: 4, 573: 4, 574: 7, 575: 4, 576: 4, 577: 4, 578: 5, 579: 7, 580: 4, 581: 7, 582: 7, 583: 7, 584: 7, 585: 7, 586: 7, 587: 7, 588: 7, 589: 7, 590: 9, 591: 9, 592: 7, 593: 7, 594: 7, 595: 7, 596: 9, 597: 9, 598: 5, 599: 5, 600: 5, 601: 5, 602: 5, 603: 5, 604: 5, 605: 5, 606: 5, 607: 5, 608: 5, 609: 5, 610: 5, 611: 5, 612: 5, 613: 5, 614: 5, 615: 5, 616: 5, 617: 5, 618: 5, 619: 5, 620: 11, 621: 11, 622: 11, 623: 11, 624: 11, 625: 11, 626: 11, 627: 11, 628: 11, 629: 11, 630: 11, 631: 11, 632: 0, 633: 0, 634: 0, 635: 0, 636: 0, 637: 0}

Modularidad calculada: 0.7532

Partición por módulos del grafo con umbral 0.09  
Modularidad: 0.7532



```
[ ]: # Justificación
# Tuve que usar un umbral de 0.1 para que saliera la modularidad, en este caso
# utilicé el algoritmo de Louvain para maximizar la modularidad al agrupar los ↵nodos
# más conectados dentro de varias comunidades y minimizando las conexiones ↵entre comunidades.
# los nodos se organizan de manera cohesiva, en el cerebro esta modularidad ↵refleja áreas
# funcionales especializadas que colaboran dentro de cada módulo, se comunican ↵con otras áreas
# usando otras conexiones más reducidas pero específicas.
# se observan de diferentes colores los diferentes módulos
```

```
[20]: # 5. 5) Determinar el conjunto del Rich Club y discutir las implicaciones ↵anatómicas y
```

```
# funcionales de este grupo de nodos (mínimo 100 palabras).
```

```
[156]: def rich_club(G, ej5, min_degree=10):

    # grados de los nodos
    grados= dict(G.degree())

    #seleccionamos los nodos con grado >= min_degree
    rc_nodes= [node for node, degree in grados.items() if degree >= min_degree]
    print(f"Nodos del Rich Club con grado {min_degree}: {rc_nodes}")

    # verificar los nodos del rich club
    if not rc_nodes:
        print("No se encontraron nodos con grado mayor o igual a ", min_degree)

    # posiciones de los nodos
    pos= nx.get_node_attributes(G, 'pos')
    x,y,z = np.array(list(pos.values())).T

    # colores y tamaños
    node_color = ['yellow' if node in rc_nodes else 'red' for node in G.nodes()]
    node_sizes = [200 if node in rc_nodes else 50 for node in G.nodes()]

    # crear figura en 3d
    fig= plt.figure(figsize=(12,8))
    ax= fig.add_subplot(111, projection='3d')

    # graficar nodos
    ax.scatter(x, y, z, c=node_color, s=node_sizes, label='Nodos')

    # Graficar aristas
    for edge in G.edges():
        if edge[0] in pos and edge[1] in pos:
            x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
            ax.plot(x_edge, y_edge, z_edge, c='b', alpha=0.5, linewidth=0.5)

    # Configurar título
    ax.set_title(ej5)
    plt.legend(["Los nodos del rich club en amarillo, los otros de rojo"])
    plt.show()

# grafo
grafo09= generate_graph(coactmat, 0.09, coordenadas)

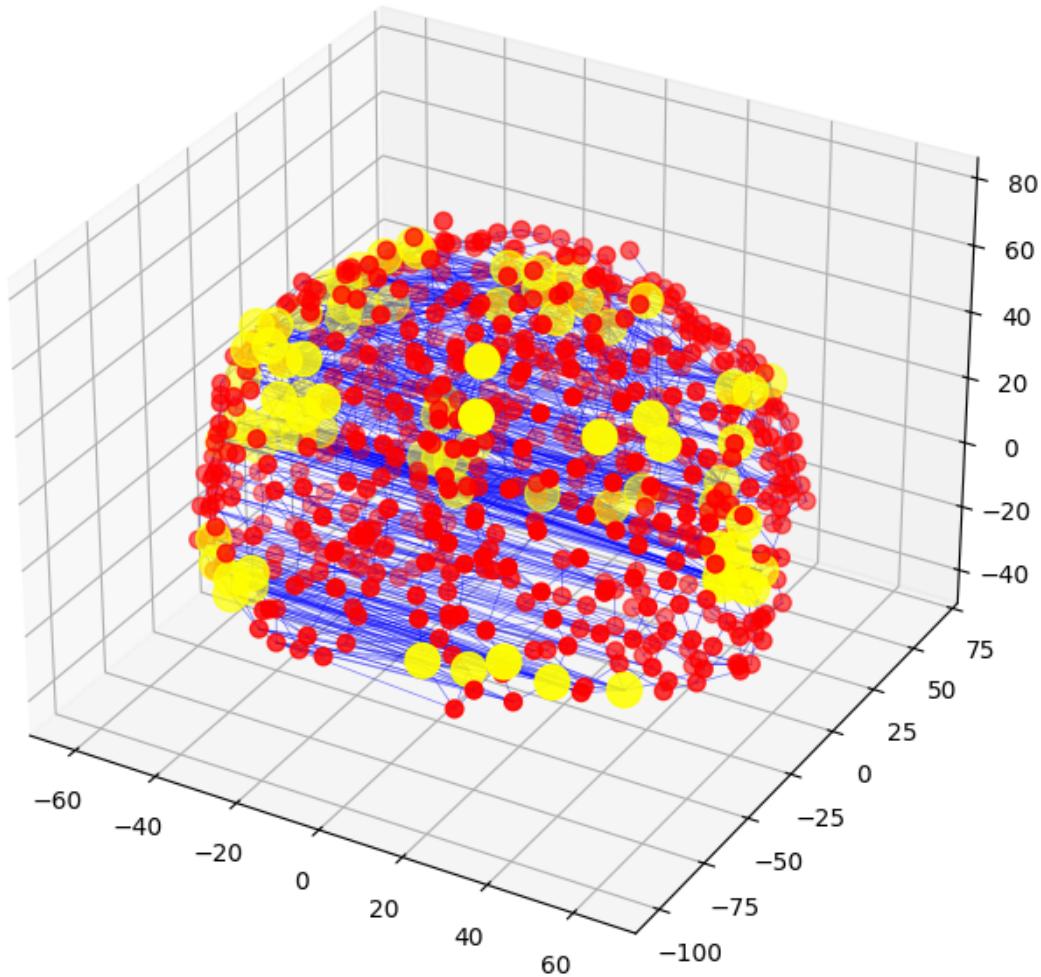
rich_club(grafo09, "Rich club con umbral 0.09", min_degree=10)
```

Nodos del Rich Club con grado 10): [19, 38, 69, 70, 100, 121, 135, 186, 202,

213, 230, 235, 237, 253, 261, 262, 266, 267, 271, 279, 285, 308, 309, 327, 328, 330, 331, 334, 344, 345, 346, 350, 352, 353, 356, 361, 362, 373, 393, 397, 400, 403, 407, 410, 416, 418, 452, 453, 454, 457, 465, 473, 477, 481, 482, 485, 488, 491, 494, 496, 500, 532, 552, 553, 599, 601, 603, 606, 607, 611, 612, 613, 615, 616, 617, 618, 619, 621, 622, 629]

Rich club con umbral 0.09

● Los nodos del rich club en amarillo, los otros de rojo



[34]: # Discusión

# El rich Club es la manera de representar muchos nodos altamente conectados, ↴ estos son cruciales para la integración global  
# de la información que recibe el cerebro. En este caso, los nodos corresponden ↴ a regiones de alto grado

```

# como la corteza prefrontal y áreas de asociación. Anatómicamente hablando, el ↵
    ↵rich club hace más fácil
# la comunicación entre los diferentes módulos cerebrales, funcionan como ↵
    ↵núcleos integradores.
# Funcionalmente hablando, que tengan tanta conectividad es importante para ↵
    ↵procesos cognitivos complejos,
# como por ejemplo la toma de decisiones, la memoria, la atención, etc.

```

[24]: # 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 pequeño y las medidas de centralidad
# (cercanía, intermediación)

```

[157]:

```

def propiedades_grafo(G):
    propiedades = {}

    # Grado medio
    avg_grad = sum(dict(G.degree()).values()) / G.number_of_nodes()
    propiedades['grado_medio'] = avg_grad

    # Coeficiente de agrupamiento (cluster)
    coef_cluster = nx.average_clustering(G)
    propiedades['coeficiente_de_agrupamiento'] = coef_cluster

    # Centralidad de cercanía (promedio)
    cercania = np.mean(list(nx.closeness_centrality(G).values()))
    propiedades['centralidad_cercania'] = cercania

    # Centralidad de intermediación (promedio)
    betweenness = nx.betweenness_centrality(G)
    intermediacion = np.mean(list(betweenness.values()))
    propiedades['centralidad_intermediacion'] = intermediacion

    return propiedades

def comparacion_propiedades(datos_originales, datos_modificados):

    print("Comparando de propiedades:")
    for prop in datos_originales.keys():
        original = datos_originales[prop]
        modificado = datos_modificados[prop]
        cambio = ((modificado - original) / original) * 100 if original != 0 ↵
    ↵else "N/A"
        print(f"[{prop.capitalize()}]:")

```

```

        print(f"  Original: {original:.4f}")
        print(f"  Modificado: {modificado:.4f}")
        print(f"  Cambio: {cambio if cambio == 'N/A' else cambio:.2f}\n")

# Generar el grafo original
grafo_original = generate_graph(coactmat, 0.1, coordenadas)
grados = dict(grafo_original.degree())

# Cálculo de las propiedades iniciales
print("Calculamos las propiedades del grafo original:")
datos_originales = propiedades_grafo(grafo_original)
print(datos_originales)

# Seleccionar nodos del Rich Club
min_degree = 10
rc_nodes = [node for node, degree in grados.items() if degree >= min_degree]
print(f"Nodos del Rich Club (grado {min_degree}): {rc_nodes}")

# Copiar el grafo y eliminar los nodos del Rich Club
grafo_modificado = grafo_original.copy()
grafo_modificado.remove_nodes_from(rc_nodes)

# Calcular las propiedades del grafo modificado
print("Calculando las propiedades del grafo sin el Rich Club:")
datos_modificados = propiedades_grafo(grafo_modificado)
print(datos_modificados)

# Comparar propiedades
comparacion_propiedades(datos_originales, datos_modificados)

# Visualizar el grafo modificado sin el Rich Club
def plot_grafo(G, titulo):
    pos = nx.get_node_attributes(G, 'pos')
    x, y, z = np.array(list(pos.values())).T

    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Graficar nodos y aristas
    scatter = ax.scatter(x, y, z, c='red', s=50)
    for edge in G.edges():
        x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
        ax.plot(x_edge, y_edge, z_edge, c='blue', alpha=0.5, linewidth=0.5)

    ax.set_title(titulo)
    plt.show()

```

```
plot_grafo(grafo_modificado, "Grafo sin el Rich Club")
```

Calculamos las propiedades del grafo original:

```
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento':  
0.24364464673564803, 'centralidad_cercania': np.float64(0.1194031264903796),  
'centralidad_intermediacion': np.float64(0.01147712930871217)}  
Nodos del Rich Club (grado 10): [38, 70, 186, 230, 235, 253, 262, 328, 330,  
334, 346, 356, 362, 397, 400, 407, 416, 454, 473, 477, 481, 482, 485, 488, 491,  
494, 532, 599, 603, 606, 607, 616, 617, 619, 629]
```

Calculando las propiedades del grafo sin el Rich Club:

```
{'grado_medio': 3.691542288557214, 'coeficiente_de_agrupamiento':  
0.22183658427439523, 'centralidad_cercania': np.float64(0.09214115900724197),  
'centralidad_intermediacion': np.float64(0.016110568269096143)}
```

Comparando de propiedades:

Grado\_medio:

Original: 4.5298

Modificado: 3.6915

Cambio: -18.51%

Coeficiente\_de\_agrupamiento:

Original: 0.2436

Modificado: 0.2218

Cambio: -8.95%

Centralidad\_cercania:

Original: 0.1194

Modificado: 0.0921

Cambio: -22.83%

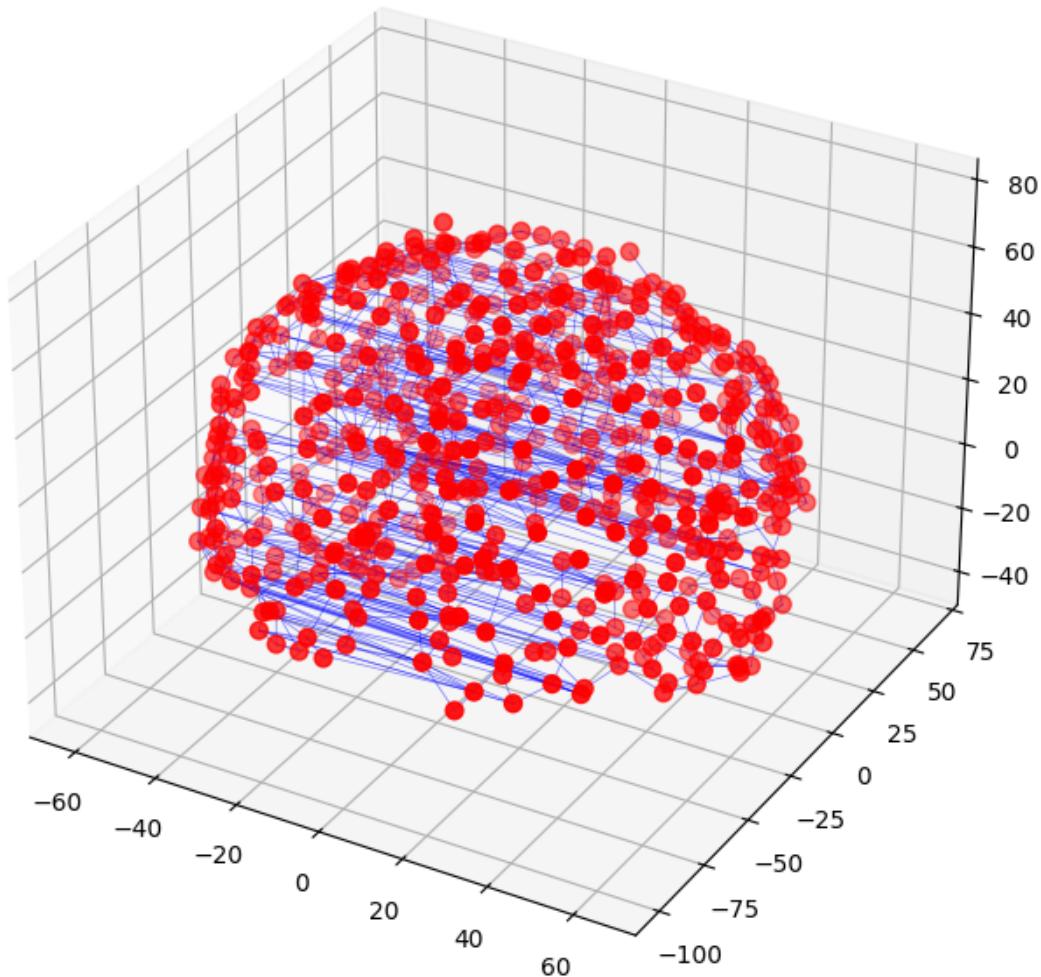
Centralidad\_intermediacion:

Original: 0.0115

Modificado: 0.0161

Cambio: 40.37%

## Grafo sin el Rich Club



```
[7]: #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  
# (cercanía, intermediación)
```

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

```

def propiedades_grafo(G):
    propiedades = {}

    # Grado medio
    avg_grad = sum(dict(G.degree()).values()) / G.number_of_nodes()
    propiedades['grado_medio'] = avg_grad

    # Coeficiente de agrupamiento (cluster)
    coef_cluster = nx.average_clustering(G)
    propiedades['coeficiente_de_agrupamiento'] = coef_cluster

    # Centralidad de cercanía (promedio)
    cercania = np.mean(list(nx.closeness_centrality(G).values()))
    propiedades['centralidad_cercania'] = cercania

    # Centralidad de intermedición (promedio)
    betweenness = nx.betweenness_centrality(G)
    intermediacion = np.mean(list(betweenness.values()))
    propiedades['centralidad_intermediacion'] = intermediacion

    return propiedades

def comparacion_propiedades(datos_originales, datos_modificados):

    print("Comparando de propiedades:")
    for prop in datos_originales.keys():
        original = datos_originales[prop]
        modificado = datos_modificados[prop]
        cambio = ((modificado - original) / original) * 100 if original != 0 else "N/A"
        print(f"{prop.capitalize()}:")
        print(f" Original: {original:.4f}")
        print(f" Modificado: {modificado:.4f}")
        print(f" Cambio: {cambio if cambio == 'N/A' else cambio:.2f}\n")

# Función para eliminar un porcentaje de nodos
def eliminar_porcentaje(G, porcentaje=0.1):

    # Calcular centralidad de intermedición
    betweenness = nx.betweenness_centrality(G)

    # Ordenar los nodos por centralidad de intermedición de mayor a menor
    sorted_nodes = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)

    # Calcular el número de nodos a eliminar
    num_nodos_eliminar = int(len(sorted_nodes) * porcentaje)
    nodos_quitar = [node for node, _ in sorted_nodes[:num_nodos_eliminar]]

```

```

# Crear una copia del grafo y eliminar los nodos
grafo_modificado = G.copy()
grafo_modificado.remove_nodes_from(nodos_quitar)

print(f"Eliminando el {percentaje*100}% de los nodos con mayor
intermediación: {nodos_quitar}"

return grafo_modificado

# Grafo original
grafo_original = generate_graph(coactmat, 0.09, coordenadas)

# Calculamos las propiedades del grafo original
print("Calculando las propiedades del grafo original: ")
datos_originales = propiedades_grafo(grafo_original)
print(datos_originales)

# Eliminar diferentes porcentajes de nodos con mayor centralidad
percentajes = [0.1, 0.2, 0.3, 0.4, 0.5]
grafo_modificado1 = eliminar_porcentaje(grafo_original,_
    ↪percentaje=percentajes[0])
grafo_modificado2 = eliminar_porcentaje(grafo_original,_
    ↪percentaje=percentajes[1])
grafo_modificado3 = eliminar_porcentaje(grafo_original,_
    ↪percentaje=percentajes[2])
grafo_modificado4 = eliminar_porcentaje(grafo_original,_
    ↪percentaje=percentajes[3])
grafo_modificado5 = eliminar_porcentaje(grafo_original,_
    ↪percentaje=percentajes[4])

# Calcular las propiedades de los nuevos grafos
print("\nCalculando las propiedades del grafo sin el 10%:")
datos_modificados_10 = propiedades_grafo(grafo_modificado1)
print(datos_modificados_10)

print("\nCalculando las propiedades del grafo sin el 20%:")
datos_modificados_20 = propiedades_grafo(grafo_modificado2)
print(datos_modificados_20)

print("\nCalculando las propiedades del grafo sin el 30%:")
datos_modificados_30 = propiedades_grafo(grafo_modificado3)
print(datos_modificados_30)

print("\nCalculando las propiedades del grafo sin el 40%:")
datos_modificados_40 = propiedades_grafo(grafo_modificado4)

```

```

print(datos_modificados_40)

print("\nCalculando las propiedades del grafo sin el 50%:")
datos_modificados_50 = propiedades_grafo(grafo_modificado5)
print(datos_modificados_50)

# Comparar propiedades
comparacion_propiedades(datos_originales, datos_modificados_10)
comparacion_propiedades(datos_originales, datos_modificados_20)
comparacion_propiedades(datos_originales, datos_modificados_30)
comparacion_propiedades(datos_originales, datos_modificados_40)
comparacion_propiedades(datos_originales, datos_modificados_50)

# Visualizar los grafos modificados
def plot_grafo(G, titulo):
    pos = nx.get_node_attributes(G, 'pos')
    x, y, z = np.array(list(pos.values())).T

    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Graficar nodos y aristas
    scatter = ax.scatter(x, y, z, c='orange', s=50)
    for edge in G.edges():
        x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
        ax.plot(x_edge, y_edge, z_edge, c='navy', alpha=0.5, linewidth=0.5)

    ax.set_title(titulo)
    plt.show()

# Graficar los grafos
plot_grafo(grafo_modificado1, "Grafo sin el 10% de nodos")
plot_grafo(grafo_modificado2, "Grafo sin el 20% de nodos")
plot_grafo(grafo_modificado3, "Grafo sin el 30% de nodos")
plot_grafo(grafo_modificado4, "Grafo sin el 40% de nodos")
plot_grafo(grafo_modificado5, "Grafo sin el 50% de nodos")

```

Calculando las propiedades del grafo original:

```

{'grado_medio': 5.614420062695925, 'coeficiente_de_agrupamiento':
0.2785052819986504, 'centralidad_cercania': np.float64(0.14410899273204028),
'centralidad_intermediacion': np.float64(0.009627134336679834)}
Eliminando el 10.0% de los nodos con mayor intermediación: [373, 235, 330, 334,
285, 619, 400, 352, 481, 275, 220, 277, 121, 353, 280, 134, 38, 50, 135, 16,
488, 482, 371, 397, 589, 618, 427, 2, 565, 591, 431, 291, 3, 0, 65, 154, 128,
344, 605, 160, 286, 517, 532, 553, 356, 237, 611, 159, 491, 410, 230, 362, 546,
328, 454, 281, 37, 513, 346, 57, 153, 69, 549]
Eliminando el 20.0% de los nodos con mayor intermediación: [373, 235, 330, 334,

```

285, 619, 400, 352, 481, 275, 220, 277, 121, 353, 280, 134, 38, 50, 135, 16, 488, 482, 371, 397, 589, 618, 427, 2, 565, 591, 431, 291, 3, 0, 65, 154, 128, 344, 605, 160, 286, 517, 532, 553, 356, 237, 611, 159, 491, 410, 230, 362, 546, 328, 454, 281, 37, 513, 346, 57, 153, 69, 549, 590, 593, 485, 195, 496, 592, 196, 55, 6, 36, 500, 329, 629, 487, 272, 579, 62, 186, 97, 582, 559, 276, 267, 223, 310, 232, 434, 193, 521, 545, 120, 22, 584, 67, 108, 253, 560, 292, 100, 399, 523, 554, 345, 599, 359, 262, 268, 61, 512, 202, 52, 318, 375, 99, 499, 279, 149, 112, 250, 111, 361, 430, 48, 18, 428, 211, 331, 259, 289, 104, 41, 327, 222, 564, 494, 407, 271, 350, 144, 516, 503, 598, 479, 363, 531, 122, 110, 116, 171, 456, 416, 552, 432, 333, 597, 226, 524, 32, 548, 562, 136, 93, 15, 283, 542, 315, 257, 417, 312, 21, 247, 563, 405, 162, 607, 204, 125, 132, 103, 438, 526, 465, 17, 339, 423, 91, 60, 209, 635, 167, 117, 585, 637, 105, 138, 302, 320, 142, 231, 194, 98, 124, 480, 522, 367, 34, 414, 514, 612, 622, 538, 248, 557, 439, 368, 303, 539, 473, 71, 210, 529, 422, 86, 306, 4, 395, 156, 70, 398, 544, 95, 73, 425, 577, 180, 123, 266, 169, 314, 54, 418, 137, 165, 452, 151, 534, 574, 406, 412, 184, 568, 596]

Eliminando el 30.0% de los nodos con mayor intermediación: [373, 235, 330, 334, 285, 619, 400, 352, 481, 275, 220, 277, 121, 353, 280, 134, 38, 50, 135, 16, 488, 482, 371, 397, 589, 618, 427, 2, 565, 591, 431, 291, 3, 0, 65, 154, 128, 344, 605, 160, 286, 517, 532, 553, 356, 237, 611, 159, 491, 410, 230, 362, 546, 328, 454, 281, 37, 513, 346, 57, 153, 69, 549, 590, 593, 485, 195, 496, 592, 196, 55, 6, 36, 500, 329, 629, 487, 272, 579, 62, 186, 97, 582, 559, 276, 267, 223, 310, 232, 434, 193, 521, 545, 120, 22, 584, 67, 108, 253, 560, 292, 100, 399, 523, 554, 345, 599, 359, 262, 268, 61, 512, 202, 52, 318, 375, 99, 499, 279, 149, 112, 250, 111, 361, 430, 48, 18, 428, 211, 331, 259, 289, 104, 41, 327, 222, 564, 494, 407, 271, 350, 144, 516, 503, 598, 479, 363, 531, 122, 110, 116, 171, 456, 416, 552, 432, 333, 597, 226, 524, 32, 548, 562, 136, 93, 15, 283, 542, 315, 257, 417, 312, 21, 247, 563, 405, 162, 607, 204, 125, 132, 103, 438, 526, 465, 17, 339, 423, 91, 60, 209, 635, 167, 117, 585, 637, 105, 138, 302, 320, 142, 231, 194, 98, 124, 480, 522, 367, 34, 414, 514, 612, 622, 538, 248, 557, 439, 368, 303, 539, 473, 71, 210, 529, 422, 86, 306, 4, 395, 156, 70, 398, 544, 95, 73, 425, 577, 180, 123, 266, 169, 314, 54, 418, 137, 165, 452, 151, 534, 574, 406, 412, 184, 568, 596]

Eliminando el 40.0% de los nodos con mayor intermediación: [373, 235, 330, 334, 285, 619, 400, 352, 481, 275, 220, 277, 121, 353, 280, 134, 38, 50, 135, 16, 488, 482, 371, 397, 589, 618, 427, 2, 565, 591, 431, 291, 3, 0, 65, 154, 128, 344, 605, 160, 286, 517, 532, 553, 356, 237, 611, 159, 491, 410, 230, 362, 546, 328, 454, 281, 37, 513, 346, 57, 153, 69, 549, 590, 593, 485, 195, 496, 592, 196, 55, 6, 36, 500, 329, 629, 487, 272, 579, 62, 186, 97, 582, 559, 276, 267, 223, 310, 232, 434, 193, 521, 545, 120, 22, 584, 67, 108, 253, 560, 292, 100, 399, 523, 554, 345, 599, 359, 262, 268, 61, 512, 202, 52, 318, 375, 99, 499, 279, 149, 112, 250, 111, 361, 430, 48, 18, 428, 211, 331, 259, 289, 104, 41, 327, 222, 564, 494, 407, 271, 350, 144, 516, 503, 598, 479, 363, 531, 122, 110, 116, 171, 456, 416, 552, 432, 333, 597, 226, 524, 32, 548, 562, 136, 93, 15, 283, 542, 315, 257, 417, 312, 21, 247, 563, 405, 162, 607, 204, 125, 132, 103, 438, 526, 465, 17, 339, 423, 91, 60, 209, 635, 167, 117, 585, 637, 105, 138, 302, 320, 142, 231, 194, 98, 124, 480, 522, 367, 34, 414, 514, 612, 622, 538, 248, 557, 439, 368, 303, 539, 473, 71, 210, 529, 422, 86, 306, 4, 395, 156, 70, 398, 544, 95, 73, 425, 577, 180, 123, 266, 169, 314, 54, 418, 137, 165, 452, 151, 534, 574, 406, 412, 184, 568, 596]

Eliminando el 50.0% de los nodos con mayor intermediación: [373, 235, 330, 334, 285, 619, 400, 352, 481, 275, 220, 277, 121, 353, 280, 134, 38, 50, 135, 16, 488, 482, 371, 397, 589, 618, 427, 2, 565, 591, 431, 291, 3, 0, 65, 154, 128, 344, 605, 160, 286, 517, 532, 553, 356, 237, 611, 159, 491, 410, 230, 362, 546, 328, 454, 281, 37, 513, 346, 57, 153, 69, 549, 590, 593, 485, 195, 496, 592, 196, 55, 6, 36, 500, 329, 629, 487, 272, 579, 62, 186, 97, 582, 559, 276, 267, 223, 310, 232, 434, 193, 521, 545, 120, 22, 584, 67, 108, 253, 560, 292, 100, 399, 523, 554, 345, 599, 359, 262, 268, 61, 512, 202, 52, 318, 375, 99, 499, 279, 149, 112, 250, 111, 361, 430, 48, 18, 428, 211, 331, 259, 289, 104, 41, 327, 222, 564, 494, 407, 271, 350, 144, 516, 503, 598, 479, 363, 531, 122, 110,

116, 171, 456, 416, 552, 432, 333, 597, 226, 524, 32, 548, 562, 136, 93, 15, 283, 542, 315, 257, 417, 312, 21, 247, 563, 405, 162, 607, 204, 125, 132, 103, 438, 526, 465, 17, 339, 423, 91, 60, 209, 635, 167, 117, 585, 637, 105, 138, 302, 320, 142, 231, 194, 98, 124, 480, 522, 367, 34, 414, 514, 612, 622, 538, 248, 557, 439, 368, 303, 539, 473, 71, 210, 529, 422, 86, 306, 4, 395, 156, 70, 398, 544, 95, 73, 425, 577, 180, 123, 266, 169, 314, 54, 418, 137, 165, 452, 151, 534, 574, 406, 412, 184, 568, 596, 365, 550, 351, 551, 261, 436, 571, 519, 547, 206, 158, 113, 201, 580, 387, 348, 308, 263, 468, 528, 509, 448, 28, 163, 570, 11, 264, 219, 381, 197, 126, 358, 441, 588, 7, 541, 613, 66, 238, 10, 29, 578, 587, 258, 192, 227, 199, 176, 508, 85, 450, 177, 555, 556, 72, 497, 566, 444, 40, 42, 274, 244, 567, 317]

Calculando las propiedades del grafo sin el 10%:

```
{'grado_medio': 4.351304347826087, 'coeficiente_de_agrupamiento':  
0.26065096256400605, 'centralidad_cercania': np.float64(0.10502884478729942),  
'centralidad_intermediacion': np.float64(0.015025974434567928)}
```

Calculando las propiedades del grafo sin el 20%:

```
{'grado_medio': 3.675146771037182, 'coeficiente_de_agrupamiento':  
0.2424004892693738, 'centralidad_cercania': np.float64(0.07423423198025483),  
'centralidad_intermediacion': np.float64(0.01762977279616532)}
```

Calculando las propiedades del grafo sin el 30%:

```
{'grado_medio': 3.185682326621924, 'coeficiente_de_agrupamiento':  
0.2563371006324026, 'centralidad_cercania': np.float64(0.05099617054174734),  
'centralidad_intermediacion': np.float64(0.02392776778147008)}
```

Calculando las propiedades del grafo sin el 40%:

```
{'grado_medio': 2.7362924281984333, 'coeficiente_de_agrupamiento':  
0.22687222843880805, 'centralidad_cercania': np.float64(0.030301707402235555),  
'centralidad_intermediacion': np.float64(0.008450092358470775)}
```

Calculando las propiedades del grafo sin el 50%:

```
{'grado_medio': 2.3510971786833856, 'coeficiente_de_agrupamiento':  
0.20209732795939692, 'centralidad_cercania': np.float64(0.020176775727589722),  
'centralidad_intermediacion': np.float64(0.0023531340530123443)}
```

Comparando de propiedades:

Grado\_medio:

Original: 5.6144

Modificado: 4.3513

Cambio: -22.50%

Coeficiente\_de\_agrupamiento:

Original: 0.2785

Modificado: 0.2607

Cambio: -6.41%

Centralidad\_cercania:

Original: 0.1441  
Modificado: 0.1050  
Cambio: -27.12%

Centralidad\_intermediacion:  
Original: 0.0096  
Modificado: 0.0150  
Cambio: 56.08%

Comparando de propiedades:  
Grado\_medio:  
Original: 5.6144  
Modificado: 3.6751  
Cambio: -34.54%

Coeficiente\_de\_agrupamiento:  
Original: 0.2785  
Modificado: 0.2424  
Cambio: -12.96%

Centralidad\_cercania:  
Original: 0.1441  
Modificado: 0.0742  
Cambio: -48.49%

Centralidad\_intermediacion:  
Original: 0.0096  
Modificado: 0.0176  
Cambio: 83.13%

Comparando de propiedades:  
Grado\_medio:  
Original: 5.6144  
Modificado: 3.1857  
Cambio: -43.26%

Coeficiente\_de\_agrupamiento:  
Original: 0.2785  
Modificado: 0.2563  
Cambio: -7.96%

Centralidad\_cercania:  
Original: 0.1441  
Modificado: 0.0510  
Cambio: -64.61%

Centralidad\_intermediacion:  
Original: 0.0096

Modificado: 0.0239  
Cambio: 148.55%

Comparando de propiedades:

Grado\_medio:  
Original: 5.6144  
Modificado: 2.7363  
Cambio: -51.26%

Coeficiente\_de\_agrupamiento:  
Original: 0.2785  
Modificado: 0.2269  
Cambio: -18.54%

Centralidad\_cercania:  
Original: 0.1441  
Modificado: 0.0303  
Cambio: -78.97%

Centralidad\_intermediacion:  
Original: 0.0096  
Modificado: 0.0085  
Cambio: -12.23%

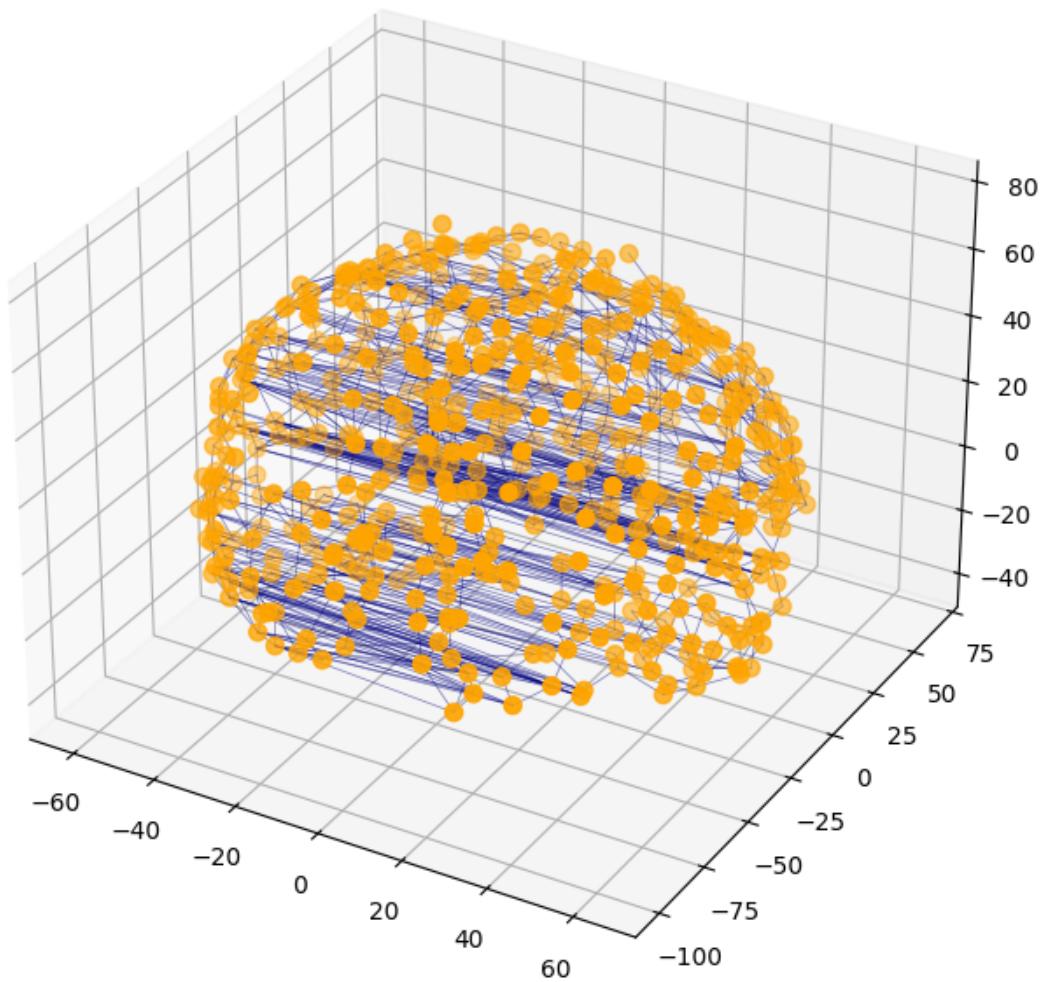
Comparando de propiedades:  
Grado\_medio:  
Original: 5.6144  
Modificado: 2.3511  
Cambio: -58.12%

Coeficiente\_de\_agrupamiento:  
Original: 0.2785  
Modificado: 0.2021  
Cambio: -27.44%

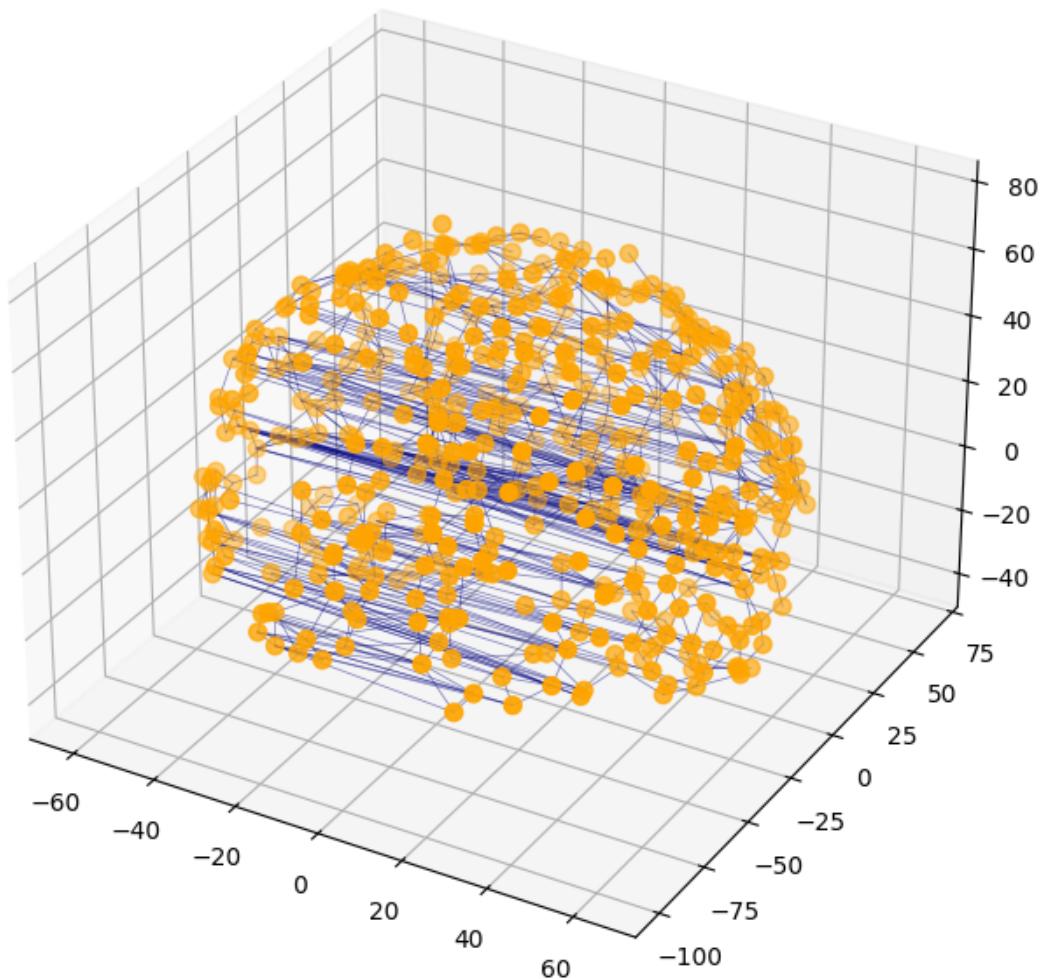
Centralidad\_cercania:  
Original: 0.1441  
Modificado: 0.0202  
Cambio: -86.00%

Centralidad\_intermediacion:  
Original: 0.0096  
Modificado: 0.0024  
Cambio: -75.56%

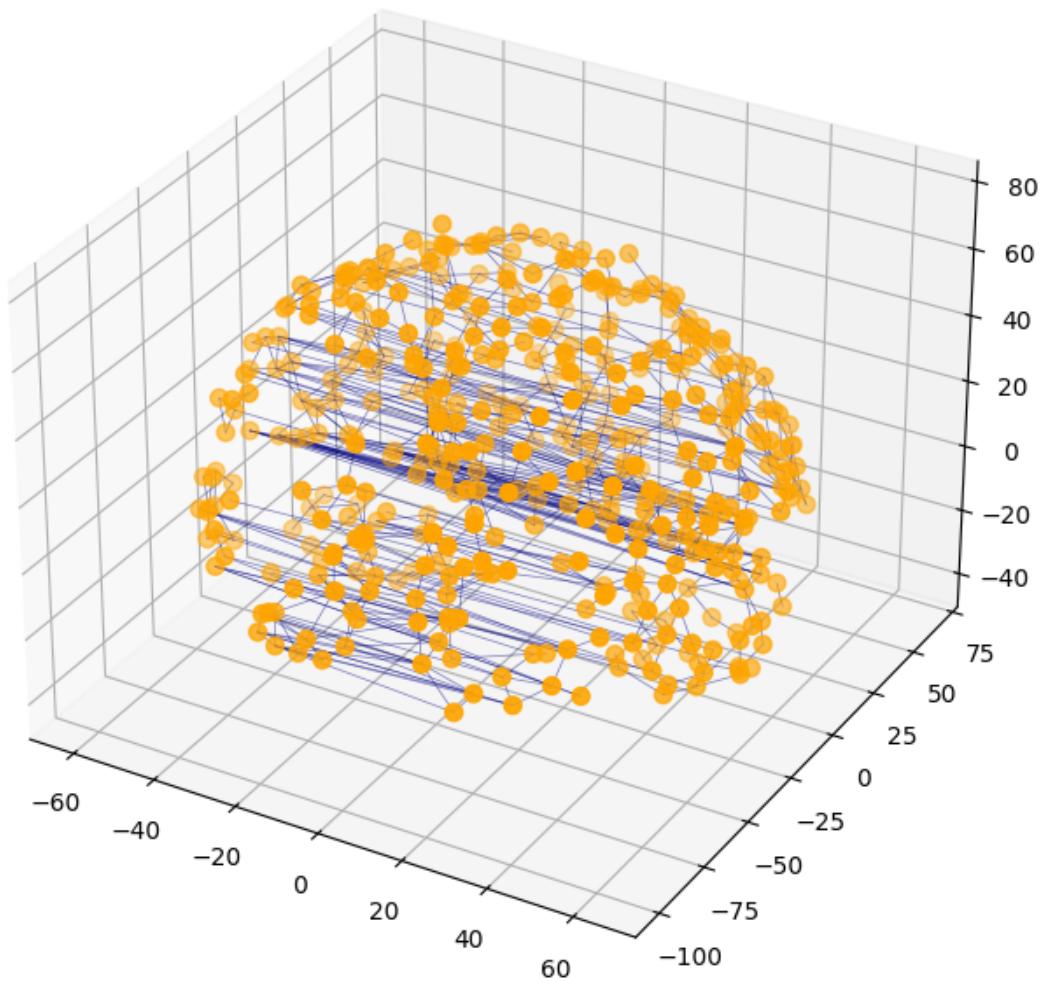
Grafo sin el 10% de nodos



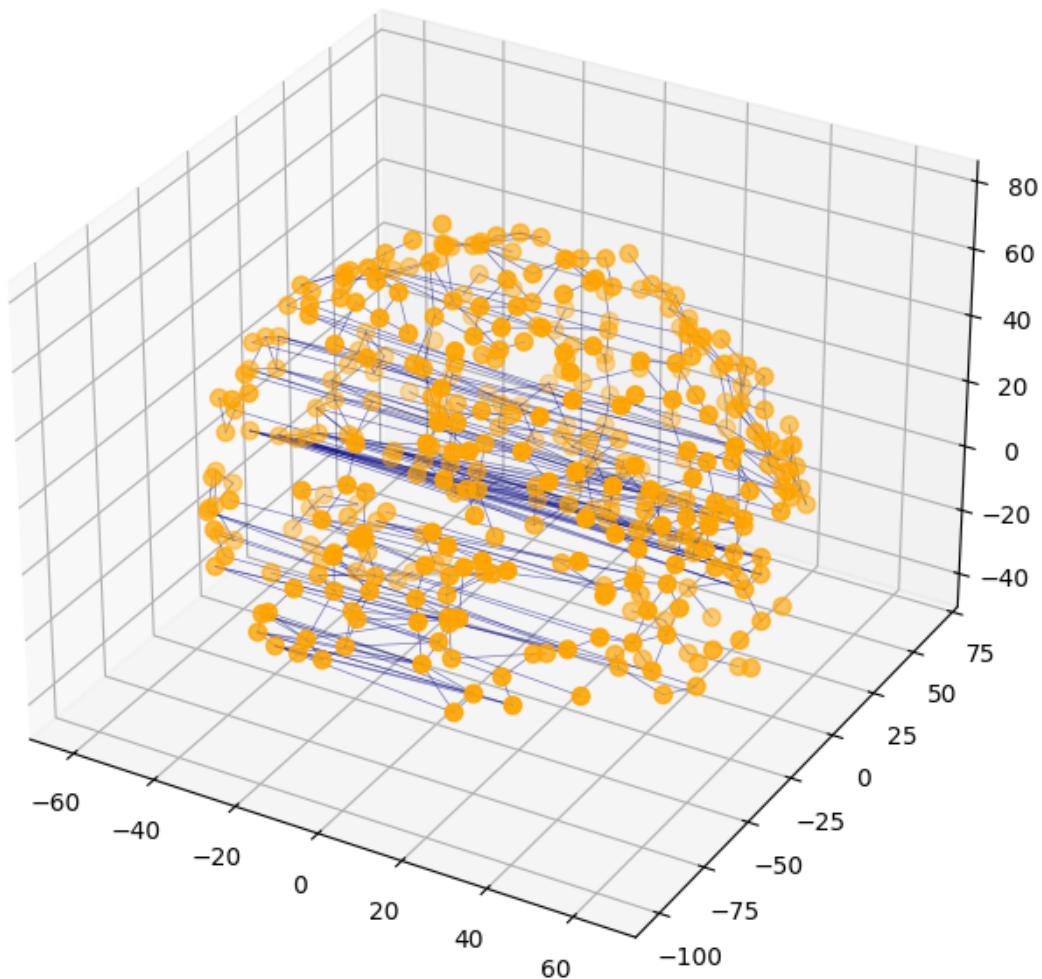
Grafo sin el 20% de nodos



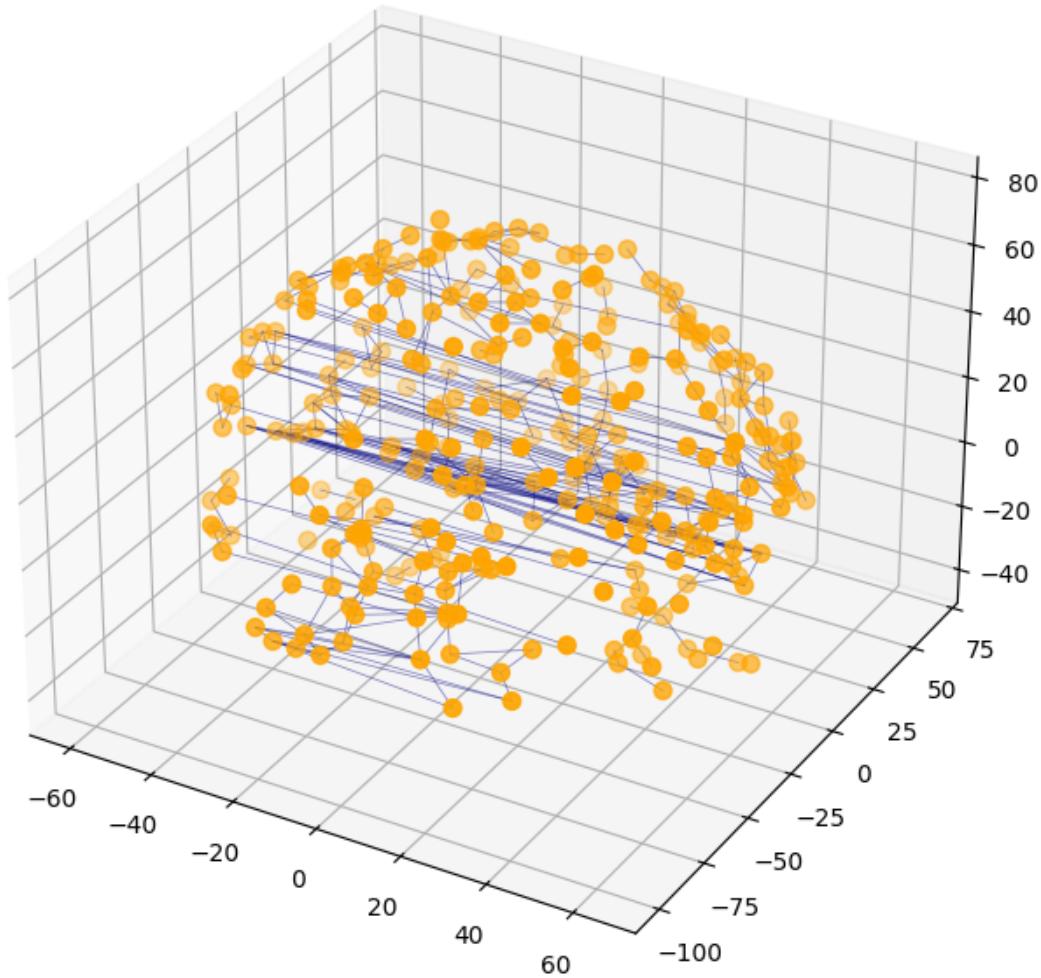
Grafo sin el 30% de nodos



Grafo sin el 40% de nodos



## Grafo sin el 50% de nodos



```
[ ]: # 8.  
# Generar un modelo nulo aleatorio donde se tenga el mismo número de nodos y el  
# mismo número total de conexiones, y comparar sus propiedades  
# con el grafo original del cerebro.
```

```
[160]: def propiedades_grafo(G):  
    propiedades = {}  
  
    # Grado medio  
    avg_grad = sum(dict(G.degree()).values()) / G.number_of_nodes()  
    propiedades['grado_medio'] = avg_grad
```

```

# Coeficiente de agrupamiento (cluster)
coef_cluster = nx.average_clustering(G)
propiedades['coeficiente_de_agrupamiento'] = coef_cluster

# Centralidad de cercanía (promedio)
cercania = np.mean(list(nx.closeness_centrality(G).values()))
propiedades['centralidad_cercania'] = cercania

# Centralidad de intermediaciación (promedio)
betweenness = nx.betweenness_centrality(G)
intermediacion = np.mean(list(betweenness.values()))
propiedades['centralidad_intermediacion'] = intermediacion

return propiedades

def comparacion_propiedades(datos_originales, datos_modificados):
    print("\nComparación de propiedades:")
    for prop in datos_originales.keys():
        original = datos_originales[prop]
        modificado = datos_modificados[prop]
        cambio = ((modificado - original) / original) * 100 if original != 0 else "N/A"
        print(f"{prop.capitalize()}:")
        print(f" Original: {original:.4f}")
        print(f" Modificado: {modificado:.4f}")
        print(f" Cambio: {cambio if cambio == 'N/A' else cambio:.2f}\n")

# Grafo original
grafo_original = generate_graph(coactmat, 0.1, coordenadas)

# propiedades del grafo original
print("Calculando las propiedades del grafo original:")
datos_originales = propiedades_grafo(grafo_original)
print(datos_originales)

# MODELO NULO ALEATORIO
num_nodos = grafo_original.number_of_nodes()
num_aristas = grafo_original.number_of_edges()

# generamos el modelo nulo con el mismo número de aristas que el original
modelo_nulo = nx.gnm_random_graph(num_nodos, num_aristas)

# calculamos las propiedades del modelo aleatorio
print("Calculando las propiedades del modelo nulo aleatorio:")
datos_nulo = propiedades_grafo(modelo_nulo)
print(datos_nulo)

```

```

# Comparar las propiedades entre el grafo original y el modelo nulo
comparacion_propiedades(datos_originales, datos_nulo)

# Visualizar el grafo original y el modelo nulo aleatorio
def plot_grafo(G, titulo):
    pos = nx.spring_layout(G) # Layout para visualizar en 2D
    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, with_labels=True, node_color='dodgerblue', node_size=50, edge_color='darkmagenta', alpha=0.5)
    plt.title(titulo)
    plt.show()

# vamos a ver si salen los grafos !!
plot_grafo(grafo_original, "Grafo Original")
plot_grafo(modelo_nulo, "Modelo Nulo Aleatorio")

```

Calculando las propiedades del grafo original:

```
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento': 0.24364464673564803, 'centralidad_cercania': np.float64(0.1194031264903796), 'centralidad_intermediacion': np.float64(0.01147712930871217)}
```

Calculando las propiedades del modelo nulo aleatorio:

```
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento': 0.00975398686997433, 'centralidad_cercania': np.float64(0.22263277191117017), 'centralidad_intermediacion': np.float64(0.005273980596966005)}
```

Comparación de propiedades:

Grado\_medio:

```
Original: 4.5298
Modificado: 4.5298
Cambio: 0.00%
```

Coeficiente\_de\_agrupamiento:

```
Original: 0.2436
Modificado: 0.0098
Cambio: -96.00%
```

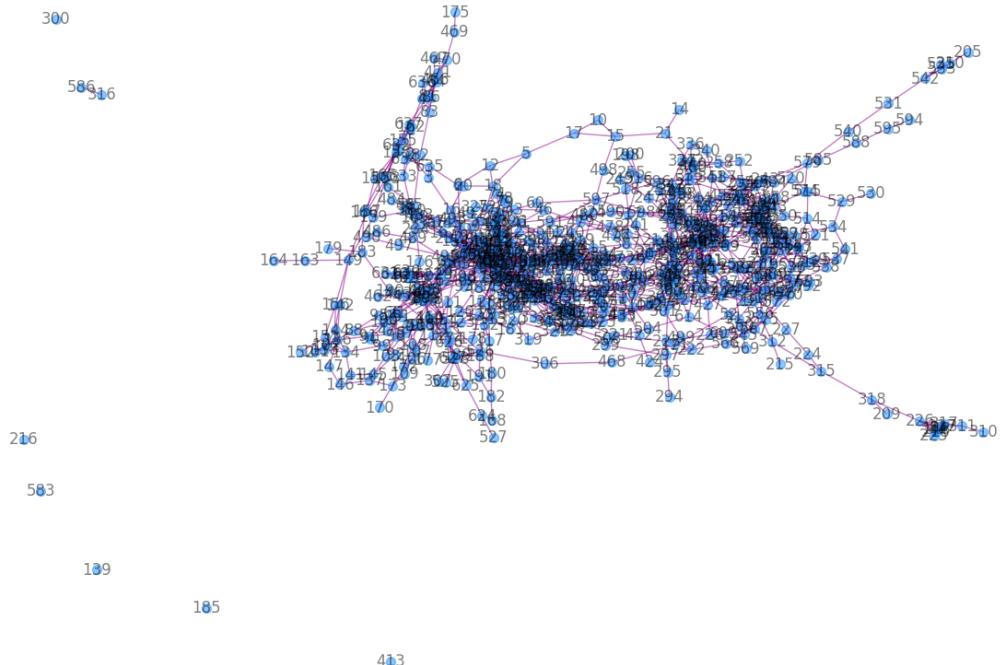
Centralidad\_cercania:

```
Original: 0.1194
Modificado: 0.2226
Cambio: 86.45%
```

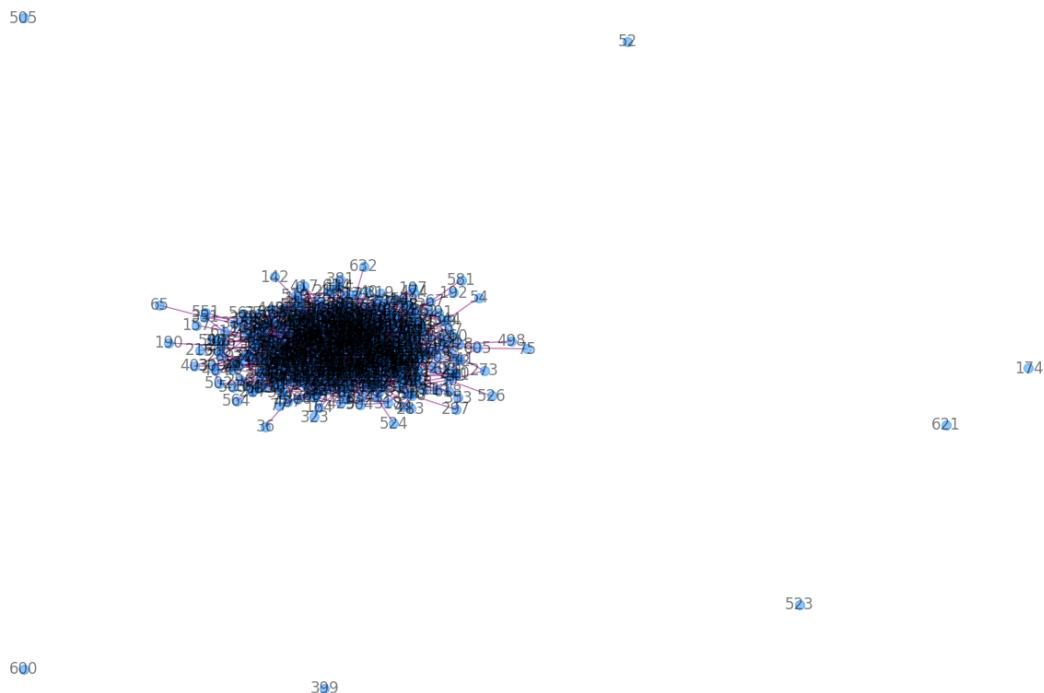
Centralidad\_intermediacion:

```
Original: 0.0115
Modificado: 0.0053
Cambio: -54.05%
```

Grafo Original



Modelo Nulo Aleatorio



```
[ ]: # 9.  
# Generar un modelo nulo aleatorio donde se conserve la distribución de grado y  
# comparar sus propiedades con el grafo original del cerebro.
```

```
[161]: import networkx as nx  
import numpy as np  
import matplotlib.pyplot as plt  
  
def propiedades_grafo(G):  
    propiedades = {}  
  
    # Grado medio  
    avg_grad = sum(dict(G.degree()).values()) / G.number_of_nodes()  
    propiedades['grado_medio'] = avg_grad  
  
    # Coeficiente de cluster  
    coef_cluster = nx.average_clustering(G)  
    propiedades['coeficiente_de_agrupamiento'] = coef_cluster  
  
    # Centralidad de cercanía  
    cercania = np.mean(list(nx.closeness_centrality(G).values()))  
    propiedades['centralidad_cercania'] = cercania  
  
    # Centralidad de intermedición  
    betweenness = nx.betweenness_centrality(G)  
    intermedacion = np.mean(list(betweenness.values()))  
    propiedades['centralidad_intermediacion'] = intermedacion  
  
    return propiedades  
  
def comparacion_propiedades(datos_originales, datos_modificados):  
  
    print("Comparamos propiedades:")  
    for prop in datos_originales.keys():  
        original = datos_originales[prop]  
        modificado = datos_modificados[prop]  
        cambio = ((modificado - original) / original) * 100 if original != 0  
    ↪else "N/A"  
        print(f"{prop.capitalize()}:")  
        print(f" Original: {original:.4f}")  
        print(f" Modificado: {modificado:.4f}")  
        print(f" Cambio: {cambio if cambio == 'N/A' else cambio:.2f}\n")  
  
    # Grafo original (usamos el grafo previamente generado)  
    grafo_original = generate_graph(coactmat, 0.1, coordenadas)
```

```

# Calcular las propiedades del grafo original
print("Calculando las propiedades del grafo original:")
datos_originales = propiedades_grafo(grafo_original)
print(datos_originales)

# MODELO NULO ALEATORIO con la misma distribución de grados
modelo_nulo = nx.configuration_model([grafo_original.degree(n) for n in
                                         grafo_original.nodes()])

# Convertir el modelo nulo en un grafo simple
modelo_nulo_multigrafo = nx.configuration_model([grafo_original.degree(n) for n in
                                                   grafo_original.nodes()])

#eliminar self-loops
modelo_nulo_multigrafo.remove_edges_from(nx.
                                         selfloop_edges(modelo_nulo_multigrafo))

# convertimos el modelo nulo en un grafo simple
modelo_nulo = nx.Graph(modelo_nulo_multigrafo)

# Calculamos las propiedades del modelo nulo aleatorio
print("Calculamos las propiedades del modelo nulo aleatorio:")
datos_nulo = propiedades_grafo(modelo_nulo)
print(datos_nulo)

# Comparar las propiedades entre el grafo original y el modelo nulo
comparacion_propiedades(datos_originales, datos_nulo)

# Visualizar el grafo original y el modelo nulo aleatorio
def plot_grafo(G, titulo):
    pos = nx.spring_layout(G)
    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, with_labels=True, node_color='red', node_size=50, edge_color='blue', alpha=0.5)
    plt.title(titulo)
    plt.show()

# Graficar el grafo original y el modelo nulo
plot_grafo(grafo_original, "Grafo Original")
plot_grafo(modelo_nulo, "Modelo Nulo Aleatorio")

```

Calculando las propiedades del grafo original:

```
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento': 0.24364464673564803, 'centralidad_cercania': np.float64(0.1194031264903796), 'centralidad_intermediacion': np.float64(0.01147712930871217)}
```

Calculamos las propiedades del modelo nulo aleatorio:

```
{'grado_medio': 4.5141065830721, 'coeficiente_de_agrupamiento':  
0.009658051754216239, 'centralidad_cercania': np.float64(0.2299546907585896),  
'centralidad_intermediacion': np.float64(0.005029043206383108)}  
Comparamos propiedades:
```

Comparamos propiedades:

rado\_medio:

Original: 4.5298

Modificado: 4.5

Coeficiente de agrupamiento:

Original: 0 2436

Modificado: 0 0097

Cambio: -96,04%

Centralidad cercanía:

Original: 0.1194

Modificado: 0.2300

Cambio: 92.59%

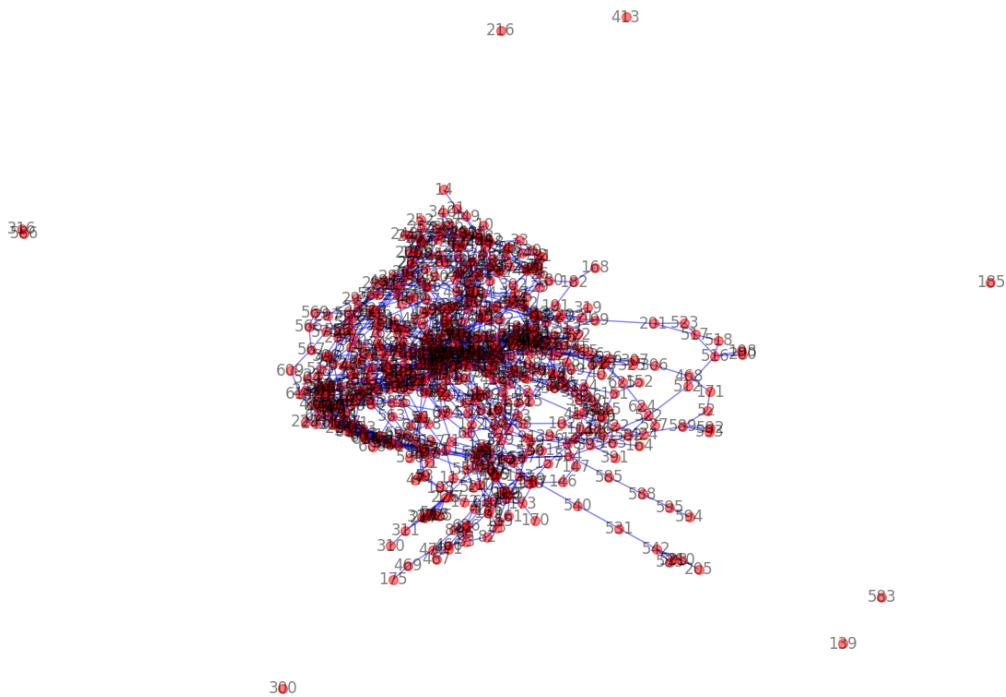
## Centralidad\_intermediacion:

Original: 0.0115

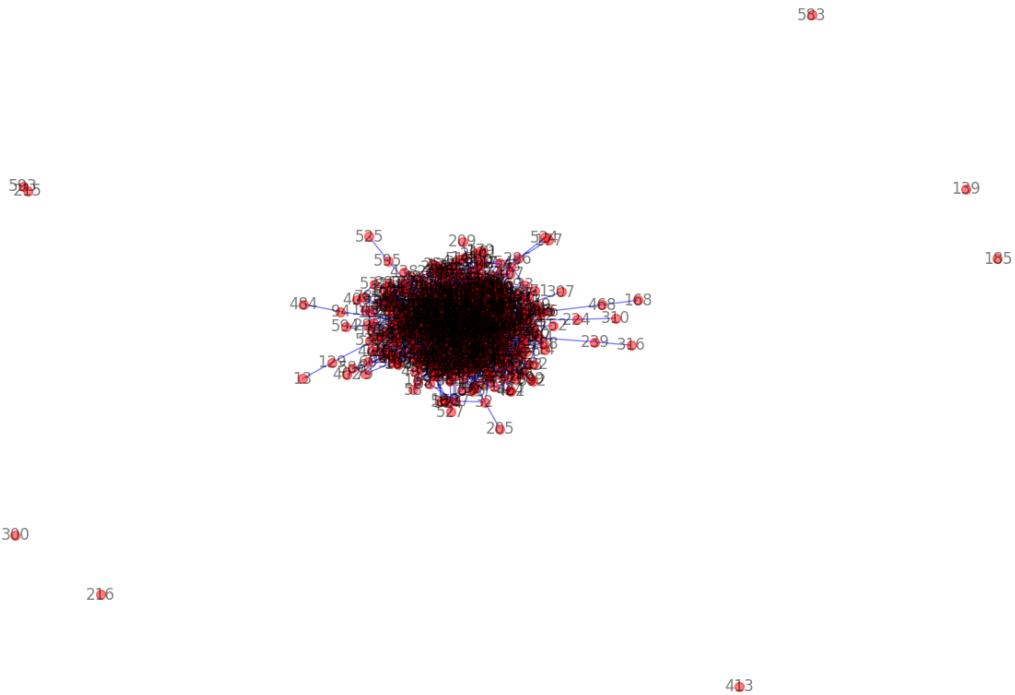
Modificado: 0.0050

Cambio: -56.18%

Grafo Original



Modelo Nulo Aleatorio



```
[ ]: # 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.
```

```
[162]: import networkx as nx  
import numpy as np  
import matplotlib.pyplot as plt  
  
def propiedades_grafo(G):  
    propiedades = {}  
  
    # Grado medio  
    avg_grad = sum(dict(G.degree()).values()) / G.number_of_nodes()  
    propiedades['grado_medio'] = avg_grad  
  
    # Coeficiente de cluster  
    coef_cluster = nx.average_clustering(G)  
    propiedades['coeficiente_de_agrupamiento'] = coef_cluster
```

```

# Centralidad de cercanía
cercania = np.mean(list(nx.closeness_centrality(G).values()))
propiedades['centralidad_cercania'] = cercania

# Centralidad de intermediación
betweenness = nx.betweenness_centrality(G)
intermediacion = np.mean(list(betweenness.values()))
propiedades['centralidad_intermediacion'] = intermediacion

return propiedades

def comparacion_propiedades(datos_originales, datos_modificados):

    print("Comparando de propiedades:")
    for prop in datos_originales.keys():
        original = datos_originales[prop]
        modificado = datos_modificados[prop]
        cambio = ((modificado - original) / original) * 100 if original != 0
    ↵else "N/A"
        print(f"{prop.capitalize()}:")
        print(f" Original: {original:.4f}")
        print(f" Modificado: {modificado:.4f}")
        print(f" Cambio: {cambio if cambio == 'N/A' else cambio:.2f}\n")

# Grafo original
grafo_original = generate_graph(coactmat, 0.1, coordenadas)

# Calcular las propiedades del grafo original
print("Calculando las propiedades del grafo original:")
datos_originales = propiedades_grafo(grafo_original)
print(datos_originales)

# Obtener las coordenadas de los nodos
nodos = list(grafo_original.nodes())
pos = nx.get_node_attributes(grafo_original, 'pos')

# Crear una matriz de distancias entre nodos
distancias = np.zeros((len(nodos), len(nodos)))
for i in range(len(nodos)):
    for j in range(len(nodos)):
        if i != j:
            distancias[i, j] = np.linalg.norm(np.array(pos[i]) - np.
    ↵array(pos[j]))

# Parámetro de decaimiento para la probabilidad de conexión
alpha = 0.1

```

```

# Crear una matriz de probabilidades basada en la distancia
probabilidades = np.exp(-alpha * distancias)

# MODELO NULO en probabilidades de conexión
num_aristas = grafo_original.number_of_edges()
modelo_nulo = nx.Graph()
modelo_nulo.add_nodes_from(nodos)

# Establecer conexiones aleatorias basadas en las probabilidades
aristas = []
while len(aristas) < num_aristas:
    i, j = np.random.choice(len(nodos), size=2, replace=False)
    if (i, j) not in aristas and (j, i) not in aristas:
        if np.random.rand() < probabilidades[i, j]:
            aristas.append((i, j))

modelo_nulo.add_edges_from(aristas)

# Calculamos las propiedades del modelo nulo
print("Calculamos las propiedades del modelo nulo con probabilidad geométrica:")
datos_nulo = propiedades_grafo(modelo_nulo)
print(datos_nulo)

# Comparar las propiedades entre el grafo original y el modelo nulo
comparacion_propiedades(datos_originales, datos_nulo)

# Visualizar el grafo original y el modelo nulo
def plot_grafo(G, titulo):
    pos = nx.spring_layout(G) # Layout para visualizar en 2D
    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, with_labels=True, node_color='orangered', node_size=50,
            edge_color='forestgreen', alpha=0.5)
    plt.title(titulo)
    plt.show()

# Graficar el grafo original y el modelo nulo
plot_grafo(grafo_original, "Grafo Original")
plot_grafo(modelo_nulo, "Modelo Nulo con Probabilidad Geométrica")

```

Calculando las propiedades del grafo original:

```
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento': 0.24364464673564803, 'centralidad_cercania': np.float64(0.1194031264903796), 'centralidad_intermediacion': np.float64(0.01147712930871217)}
```

Calculamos las propiedades del modelo nulo con probabilidad geométrica:

```
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento': 0.05488336115295363, 'centralidad_cercania': np.float64(0.1927184775258024),
```

```
'centralidad_intermediacion': np.float64(0.006522120566176706)}
```

Comparando de propiedades:

Grado\_medio:

Original: 4.5298

Modificado: 4.5298

Cambio: 0.00%

Coeficiente\_de\_agrupamiento:

Original: 0.2436

Modificado: 0.0549

Cambio: -77.47%

Centralidad\_cercania:

Original: 0.1194

Modificado: 0.1927

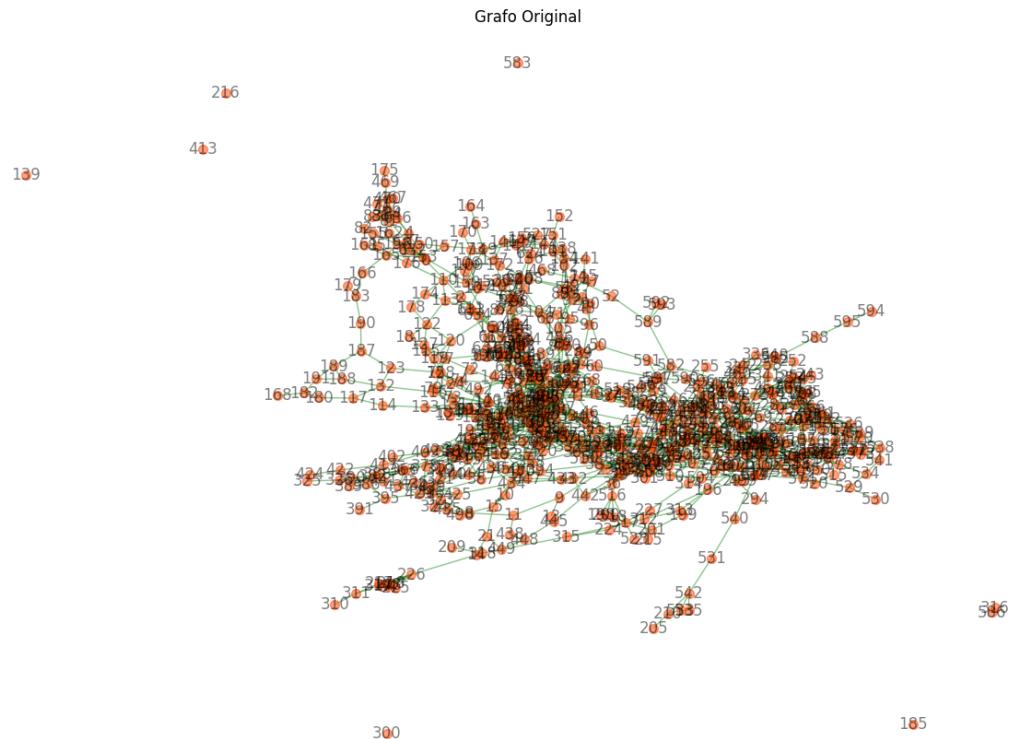
Cambio: 61.40%

Centralidad\_intermediacion:

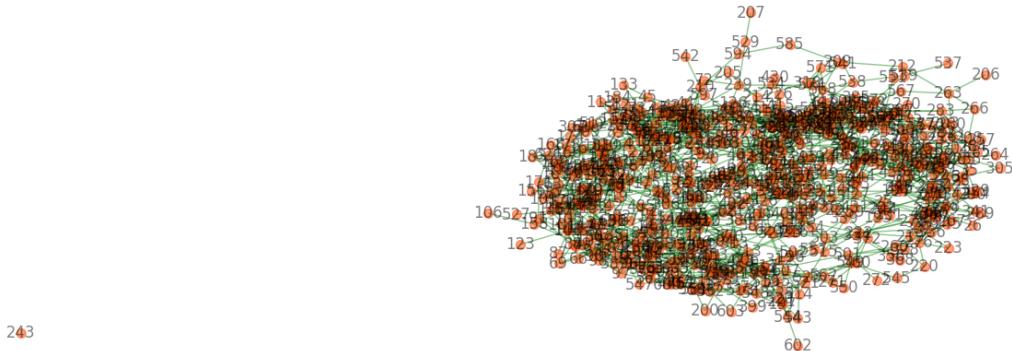
Original: 0.0115

Modificado: 0.0065

Cambio: -43.17%



## Modelo Nulo con Probabilidad Geométrica



400

151

[ ]: # Reseña sobre lo aprendido en el curso

[ ]: # Creo que aprender herramientas tan importantes como los diferentes lenguajes de programación para poder ser usados en el campo de las neurociencias me parece muy importante!!! Me costó mucho trabajo, no se me da tan fácil la programación,

# si creo que es importante enseñar bien para meterse bien en el mundo de la programación, aún más sabiendo que es un requisito

# vital e importante para nuestra carrera. Me gustó poder aprender la teoría de grafos, me gustó que las matrices que hemos aprendido

# a calcular y a graficar en clase sean utilizadas para cosas tan importantes como la visualización de datos del conectoma humano.

# Espero poder mejorar mis habilidades de programación, es pesado, pero es muy satisfactorio ver que te salen los programas.

# Creo que es importante dar a conocer más lo que hacemos en neurociencias, muchas personas que conozco (que saben de programación y que utilizan grafos) no sabían que se podían hacer cosas así en jupyter y en matlab!! Tenemos un deber grande, es importante difundir la información, # divulgar los conocimientos que generamos.