Redes complejas

---

# Práctica 5:

MUSI - Máster Universitario en Sistemas Inteligentes

López Muñoz, Iván - 43221427B

# Índice de contenidos

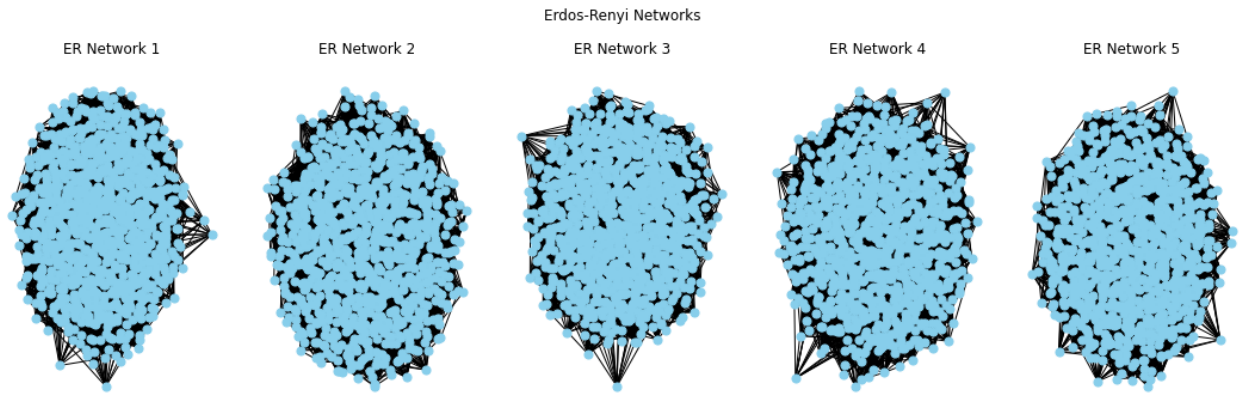
- **Seleccione una red real. Puedes usar una red utilizada en una tarea anterior.**

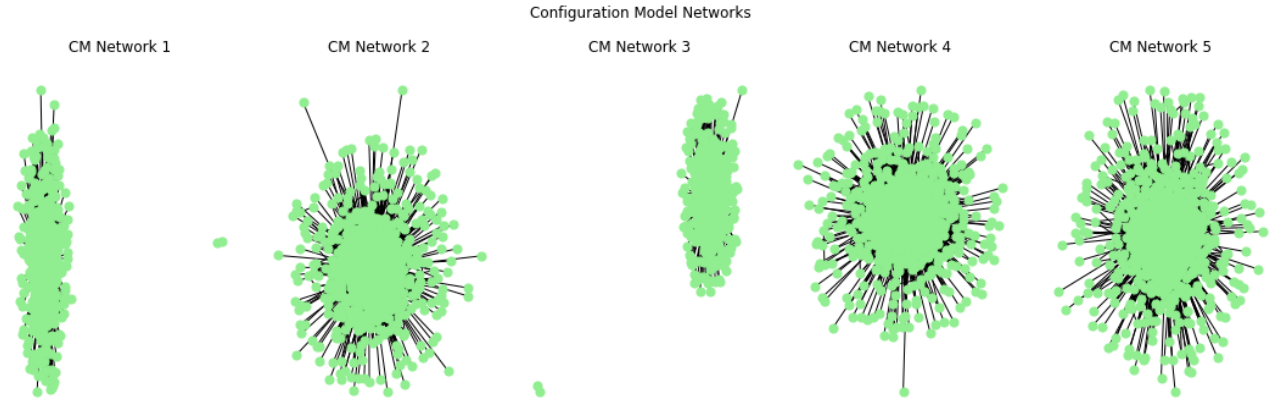
La red se generó utilizando datos de correo electrónico de una gran institución de investigación europea (https://snap.stanford.edu/data/email-Eu-core-temporal.html). Se ha anonimizado la información sobre todos los correos electrónicos entrantes y salientes entre miembros de la institución de investigación. Los correos electrónicos solo representan la comunicación entre los miembros de la institución, y el conjunto de datos no contiene mensajes entrantes ni salientes hacia el resto del mundo. Una arista dirigida (u, v) significa que la persona u envió un correo electrónico a la persona v. Se crea una arista independiente para cada destinatario del correo electrónico. De esta forma se crea un único componente en toda la red:

Number of nodes: 986
Number of edges: 16064

1. **Genera 5 redes aleatorias para cada uno de los modelos (Erdos-Renyi y Configuración) con los ¨”mismos” parámetros que la red real. Comenta estos parámetros en el texto.**

Se obtienen las diferentes redes para cada uno de los modelos:






Para generar redes siguiendo el modelo de Erdos-Renyi, se utiliza la cantidad de nodos y la probabilidad de conexiones entre ellos. Para calcular la probabilidad de conexiones entre nodos primero se calcula el número total de posibles parejas en la red:

$$N_p = \binom{N}{2} = \frac{N(N-1)}{2}$$

y como el número de parejas enlazadas por el modelo es M, se tiene por lo tanto la expresión analítica de la probabilidad de conexiones entre nodos:

$$p_c = \frac{M}{N_p} = \frac{2M}{N(N-1)}$$

Para el modelo de Configuración, se utiliza la secuencia de grados del gráfico original.

<u>Código para generar las redes aleatorias</u>

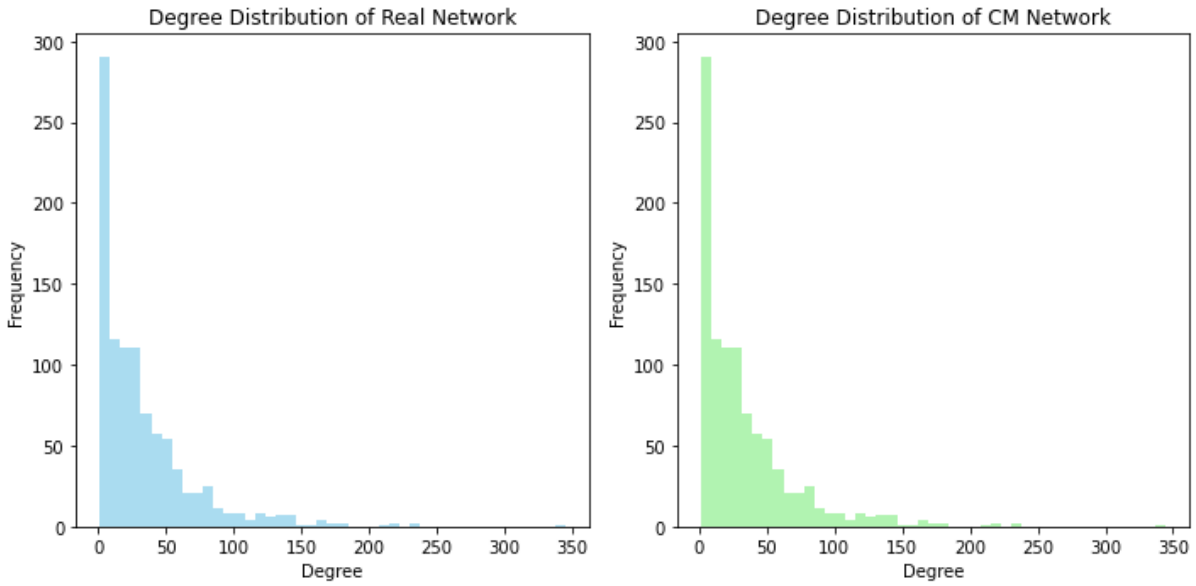
```
# Get number of nodes and edges
num_nodes = graph.number_of_nodes()
num_edges = graph.number_of_edges()

# Calculate edge probability for Erdos-Renyi model
p = 2*num_edges / (num_nodes * (num_nodes - 1)) if num_nodes > 1 else 0


# Generate 5 random ER networks with the same number of nodes and edge probability
er_networks = []
for _ in range(5):
```

```
    er_graph = nx.erdos_renyi_graph(num_nodes, p)
    er_networks.append(er_graph)

# Get degree sequence from the original graph
degree_sequence = sorted([d for n, d in graph.degree()], reverse=True)

# Generate 5 random CM networks with the same degree sequence
cm_networks = []
for _ in range(5):
    # Shuffle the degree sequence to create variations
    random.shuffle(degree_sequence)
    cm_graph = nx.configuration_model(degree_sequence)
    cm_networks.append(cm_graph)

# Plot Erdos-Renyi networks
plt.figure(figsize=(15, 5))
plt.suptitle('Erdos-Renyi Networks')

for i, er_graph in enumerate(er_networks, 1):
    plt.subplot(1, 5, i)
    nx.draw(er_graph, node_color='skyblue', node_size=50)
    plt.title(f'ER Network {i}')

plt.tight_layout()
plt.show()

# Plot Configuration Model networks
plt.figure(figsize=(15, 5))
plt.suptitle('Configuration Model Networks')

for i, cm_graph in enumerate(cm_networks, 1):
    plt.subplot(1, 5, i)
    nx.draw(cm_graph, node_color='lightgreen', node_size=50)
    plt.title(f'CM Network {i}')

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

2. **¿Se puede obtener una red con una distribución de grados muy similar a la red real utilizando generadores aleatorios? Explique sus resultados. Pon ejemplos e histogramas.**

El Modelo de Configuración genera redes aleatorias que reflejan la distribución de grados de redes reales al preservar el número de conexiones que tiene cada nodo. Captura la secuencia de grados de la red original, asegurando el mismo número de nodos con cada grado. Al reconfigurar aleatoriamente las conexiones manteniendo esta secuencia, se crea una nueva red que coincide con la distribución de grados de la red original. Pueden surgir pequeñas variaciones debido a la aleatoriedad.

Al realizar ambos histogramas, se obtienen la misma distribución de grados utilizando el modelo de configuración que con la red real:




<u>Código para generar los histogramas</u>

```
# Get the degree sequence from the real network
degree_sequence_real = [d for n, d in graph.degree()]

# Generate a Configuration Model network with the same degree sequence
cm_graph = nx.configuration_model(degree_sequence_real)
```

```
# Calculate degree sequence of the CM network
degree_sequence_cm = [d for n, d in cm_graph.degree()]

# Plot histograms to compare degree distributions
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.hist(degree_sequence_real, bins='auto', color='skyblue', alpha=0.7)
plt.title('Degree Distribution of Real Network')
plt.xlabel('Degree')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.hist(degree_sequence_cm, bins='auto', color='lightgreen', alpha=0.7)
plt.title('Degree Distribution of CM Network')
plt.xlabel('Degree')
plt.ylabel('Frequency')

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

3. **Verifica si la paradoja de la amistad se cumple en la red real y compara sus resultados con las redes generadas aleatoriamente calculadas en 1.**

La paradoja de la amistad afirma que en la mayoría de las redes sociales, tus amigos tienden a tener más amigos que tú en promedio. Para verificar esto en una red determinada, necesitamos comparar el grado promedio de los nodos con el grado promedio de sus vecinos:

```
Real Network:
Average Degree of Nodes: 32.5841784989858
Average Degree of Neighbors: 60.91982071713147

Erdos-Renyi Network:
Average Degree of Nodes: 32.81947261663286
Average Degree of Neighbors: 34.039369592088995
```

Configuration Model Network:
Average Degree of Nodes: 32.5841784989858
Average Degree of Neighbors: 60.88539591633466

La red real demuestra la paradoja de la amistad, donde las personas tienden a tener menos conexiones (32.58) que sus amigos (en promedio, 60.92).

En cambio, el modelo de Erdös-Rényi, aunque tiene un grado de nodos promedio similar, no muestra fuertemente la paradoja, ya que el promedio de grados de los vecinos (34.04) está cerca del grado de nodos promedio.

Por último el Modelo de Configuración imita de cerca la red real, mostrando una diferencia significativa entre el grado de nodos promedio y el grado de los vecinos, lo que indica que la paradoja se cumple en esta red aleatoria
.
La discrepancia marcada entre el grado de nodos promedio y el promedio de grado de los vecinos en la red real y en la red del Modelo de Configuración respalda la existencia de la paradoja en estas redes, mientras que es menos pronunciado en el modelo de Erdös-Rényi.

Código para calcular los grados promedio de cada red

```
# Calculate average degree of nodes and their neighbors in the real network
avg_degree_real = sum(dict(graph.degree()).values()) / len(graph.nodes())

# Calculate average degree of neighbors
avg_neighbor_degree_real = sum(graph.degree(nbr) for n, nbr in graph.edges()) /
len(graph.edges())

print("Real Network:")
print(f"Average Degree of Nodes: {avg_degree_real}")
print(f"Average Degree of Neighbors: {avg_neighbor_degree_real}\n")

# Generate Erdos-Renyi network with the same number of nodes and edge probability as
the real network
num_nodes = graph.number_of_nodes()
num_edges = graph.number_of_edges()
```

```
p = num_edges / (num_nodes * (num_nodes - 1) / 2) if num_nodes > 1 else 0

er_graph = nx.erdos_renyi_graph(num_nodes, p)

# Calculate average degree of nodes and their neighbors in the Erdos-Renyi network
avg_degree_er = sum(dict(er_graph.degree()).values()) / len(er_graph.nodes())
avg_neighbor_degree_er = sum(er_graph.degree(nbr) for n, nbr in er_graph.edges()) /
len(er_graph.edges())

print("Erdos-Renyi Network:")
print(f"Average Degree of Nodes: {avg_degree_er}")
print(f"Average Degree of Neighbors: {avg_neighbor_degree_er}\n")

# Generate Configuration Model network with the same degree sequence as the real
network
degree_sequence = [d for n, d in graph.degree()]
cm_graph = nx.configuration_model(degree_sequence)

# Calculate average degree of nodes and their neighbors in the Configuration Model network
avg_degree_cm = sum(dict(cm_graph.degree()).values()) / len(cm_graph.nodes())
avg_neighbor_degree_cm = sum(cm_graph.degree(nbr) for n, nbr in cm_graph.edges()) /
len(cm_graph.edges())

print("Configuration Model Network:")
print(f"Average Degree of Nodes: {avg_degree_cm}")
print(f"Average Degree of Neighbors: {avg_neighbor_degree_cm}")
```

---