Algoritmos de Redes

Informe de la Octava Práctica de Laboratorio

Detección de Comunidades en Redes y Centralidad de betweenness: Brandes y Girvan-Newman

Diego Moreno y Daniel Lagos

Grupo 13 CAIM

Universitat Politècnica de Catalunya

Diciembre 2025

Índice

Algoritmo de Brandes

Objetivos

El algoritmo de Brandes es un algoritmo de detección de comunidades mediante el análisis de la centralidad *betweenness* que, a diferencia del procedimiento Brute Force de centralidad *betweenness*, implementa el método de agregar todas las contribuciones objetivo en un solo paso de dependencias. Este algoritmo presenta un coste de complejidad temporal de $O(|V|*|E|)$, a diferencia del brute force, $\Omega(|V|^2*|E|)$.

Completa el código del enunciado para implementar el algoritmo de Brandes. Comprueba su precisión y eficiencia en la base de datos *email-Eu-core*.

Procedimiento y Resultados

Hemos diseñado un código que calcula el betweenness de todas las aristas y escribe por terminal la comparativa respecto al resultado de NetworkX:

A continuación adjuntamos nuestra implementación del algoritmo de Brandes y un output con las 10 aristas con el betweenness más significativo:

```
def edge_betweenness(G):
    bet = defaultdict(float)
    V = list(G.nodes())

    for s in V:
        pred = {v: [] for v in V}
        sigma = {v: 0 for v in V}
        dist = {v: -1 for v in V}
        delta = {v: 0 for v in V}

        sigma[s] = 1
        dist[s]  = 0

        queue = deque([s])
        stack = []

        while queue:
            v = queue.popleft()
            stack.append(v)
```

```
            for w in G.neighbors(v):
                if dist[w] < 0:
                    queue.append(w)
                    dist[w] = dist[v] + 1

                if dist[w] == dist[v] + 1:
                    sigma[w] += sigma[v]
                    pred[w].append(v)

        while stack:
            w = stack.pop()
            for v in pred[w]:
                if sigma[w] == 0: continue
                c = (sigma[v] / sigma[w]) * (1.0 + delta[w])
                bet[normalize_edge((v, w))] += c
                delta[v] += c

    return bet
```

Esta implementación itera sobre cada nodo usándolo como punto de partida y ejecuta dos fases: primero un BFS hacia adelante que calcula la distancia mínima y cuenta cuántos caminos óptimos existen hacia cada nodo (sigma) guardando el orden de visita en una pila y segundo, un recorrido hacia atrás que desapila los nodos desde el más lejano calculando la "dependencia" o flujo de tráfico (delta) y repartiéndolo proporcionalmente entre las aristas que conectan con sus predecesores según la fórmula matemática de Brandes acumulando finalmente este valor en el contador total de cada arista.

aresta | bet | NX | diff
(62, 498) | 1166.0 | 1166.0 | 0.0000
(498, np.int64(538)) | 1009.4 | 1009.4 | 0.0000
(np.int64(4), 712) | 992.0 | 992.0 | 0.0000
(np.int64(72), 106) | 788.2 | 788.2 | 0.0000
(np.int64(62), 666) | 658.3 | 658.3 | 0.0000
(np.int64(4), 62) | 581.2 | 581.2 | 0.0000
(np.int64(62), np.int64(521)) | 546.1 | 546.1 | 0.0000
(np.int64(62), 377) | 540.8 | 540.8 | 0.0000
(62, 258) | 528.5 | 528.5 | 0.0000
(96, 728) | 512.9 | 512.9 | 0.0000

No diff

Podemos observar con este output que la implementación es correcta (o al menos coincide en

el top 10 de aristas) al ser comparada con la implementación en NetworkX, devolviendo una diferencia de 0 en todos los casos.

Girvan-Newman

Objetivos

El algoritmo Girvan-Newman es otro algoritmo de detección de comunidades mediante la comparación de centralidad betweenness mediante la iteración de particiones cada vez más granulares del grafo hasta que todos los nodos hayan sido aislados.

Completa el código del enunciado para implementar el algoritmo de Givan-Newman en la red *smallG*.

- Rastrea la puntuación de modularidad (Q) en cada iteración. ¿Cómo evoluciona Q mientras progresa el algoritmo?
- Compara el algoritmo Girvan-Newman al algoritmo de Louvain en la misma base de datos. ¿Cuál de los dos algoritmos detecta una mayor Q?
- Compara el clustering detectado por el algoritmo Girvan-Newman y el algoritmo de Louvain con la verdad fundamental. En específico, compara el número de clusters, la exhaustividad (el *recall*) de los pares de vértices, y/o el índice de Rand ajustado.

Procedimiento y Resultados

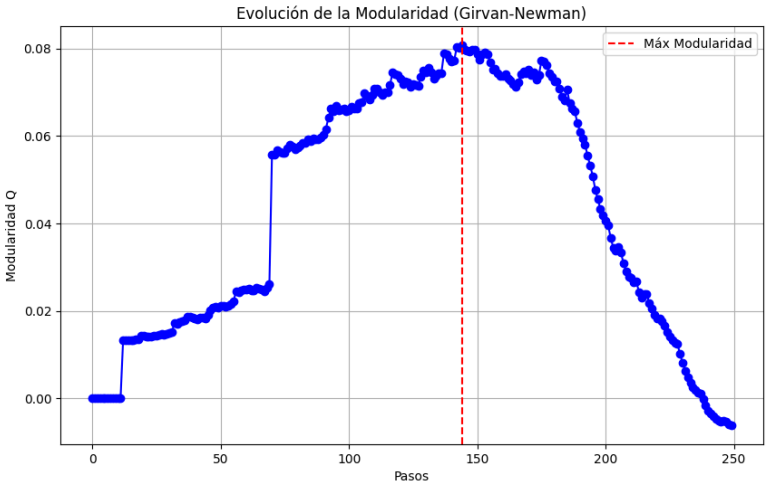
Hemos creado un código que además de calcular la modularidad en cada iteración, guarda los datos y crea una gràfica para que sea más fàcil de visualizar. Además compararemos los resultados con la implementación de Louvain implementada en NetworkX:

```
def girvan_newman(G):
    G0 = G.copy()
    G = G.copy()

    partitions = []
    modularities = []

    initial_comps = list(nx.connected_components(G))
    partitions.append(initial_comps)
    modularities.append(compute_modularity(G0, initial_comps))

    while G.number_of_edges() > 0:
```

```
        bet = edge_betweenness(G)

        if not bet: break
        edge_to_remove = max(bet.items(), key=lambda x: x[1])[0]
        G.remove_edge(*edge_to_remove)

        comps = list(nx.connected_components(G))

        if len(comps) > len(partitions[-1]):
            mod = compute_modularity(G0, comps)
            partitions.append(comps)
            modularities.append(mod)
            print(f"Comunidades: {len(comps)}, Modularidad: {mod:.4f}")

    return partitions, modularities
```

Nuestro código crea dos copias del grafo (una para modificar y otra estática para medir la calidad) y entra en un bucle donde calcula repetidamente la betweenness de todas las aristas presentes, identifica la que tiene el valor más alto (la que soporta más tráfico entre zonas) y la elimina. Tras cada eliminación verifica si el grafo se ha roto en más componentes conexas. Si ha surgido una nueva comunidad guarda la partición resultante y calcula su modularidad Q basándose en el grafo original devolviendo al final el historial de todas las divisiones y sus puntuaciones para poder elegir la mejor configuración.

A continuación adjuntamos los resultados de la ejecución sobre la base de datos proporcionada en el enunciado:



metode	mod (Q)	num comunitats	ARI vs REAL
GN	0.0808	145	0.0381
Louvain	0.3165	6	0.2913
realitat (GT)	0.1924	34	1.0000

El análisis revela que el método Girvan-Newman (GN) tiene un desempeño deficiente al intentar recuperar la estructura real de la red. Aunque la gráfica muestra que el algoritmo maximiza su modularidad en el paso 145, este pico es muy bajo ($Q \approx 0.08$) y resulta en una sobre-segmentación extrema detectando 145 comunidades frente a las 34 reales, lo que se confirma con un índice ARI casi nulo (0.0381). Por otro lado el método Louvain logra la modularidad más alta (0.3165) superando incluso la modularidad real (0.1924) pero lo hace a costa de agrupar excesivamente los nodos en solo 6 comunidades. Esto indica que Louvain encuentra una estructura matemática "fuerte" pero que no coincide con la realidad aunque su ARI (0.2913) es significativamente mejor que el de GN.